UNIVERSIDADE ESTADUAL PAULISTA

FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Treinamento de uma Rede Neural de Base Radial usando Computação Evolutiva: Implementação e Aplicações

Dissertação apresentada à Comissão de Pós-Graduação da Faculdade de Engenharia de Ilha Solteira, como requisito para a obtenção do título de Mestre em Engenharia Elétrica

Autor: **Renato Nagashima**

Orientador: **Nobuo Oki**

Ilha Solteira, 13 de Janeiro de 2006

Aos meus pais, pelo apoio,
carinho e dedicação.

Agradecimentos

Ao meu orientador, Prof. Dr. Nobuo Oki pela sua incessante paciência e dedicação ao longo deste trabalho e também pela oportunidade de aprimoramento dos meus conhecimentos.

Aos Professores do curso de Pós-graduação em Engenharia Elétrica, pela contribuição em minha formação acadêmica.

Ao Prof. Carlos Alves pela sua enorme paciência por me ajudar com o projeto na parte de programação.

Aos técnicos do laboratório da Engenharia Elétrica pelo apoio que me deram na parte técnica.

Aos meus irmãos Rogério e Leandro e minha namorada Juliana por sempre me apoiarem nas horas difíceis.

E acima de tudo, agradeço a Deus por tudo que Ele vem proporcionando de bom na minha vida.

RESUMO

Este trabalho apresenta a implementação de uma Rede Neural de Base Radial (RNBR) utilizando tecnologia 0.8μm BiCMOS da *ÁustriaMicroSystems* (AMS) e seu respectivo treinamento utilizando a Computação Evolucionária (CE).

O Algoritmo Genético (AG) foi o algoritmo de treinamento utilizado, pois é de simples operação, fácil implementação, eficaz na busca da região onde, provavelmente, encontra-se o mínimo global e é aplicável em situações onde não se conhece o modelo matemático ou este é impreciso.

A finalidade deste trabalho é mostrar a capacidade de se fazer o treinamento de uma rede neural em um *hardware*, utilizando a Computação Evolucionária. Para demonstrar a viabilidade desta rede foram implementadas duas aplicações: a conversão de um sinal triangular em um sinal senoidal e a linearização de um oscilador controlado por tensão.

Os resultados experimentais obtidos mostram a viabilidade deste treinamento.

ABSTRACT

This work describes the implementation of Radial Basis Neural Netwoks (RBNN) in 0.8μm BiCMOS technology of AustriaMicroSystems (AMS) and it is training using the Evolutionary Computation.

The Genetic Algoritmic (AG) was the training algorithmic choice due its simple operation, easy implementation and efficient way to find the minimum global point. Also it can be applied when the mathematical model was not well formulated or inaccurate.

The aim of this work is show the capacity of training the neural network implemented in hardware using the Evolutionary Computation. For show the feasibility of this neural network two application were implemented: the triangular sinusoidal signal conversion and the voltage controlled oscillator linearization.

The experimental results show the feasibility of this training.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1
INTRODUÇÃO

1.1) Motivação

Este trabalho é continuação de projetos anteriores de mestrado. Lucks[1] apresentou a idéia da rede neural e Oliveira[2] mostrou como foi feito a implementação da rede com a tecnologia de circuitos integrados BiCMOS.

Neste trabalho, será dada uma breve idéia de como foram feitos os blocos de implementação da rede neural e como foram feitos os treinamentos da rede através do Algoritmo Genético (AG).

Há vários circuitos propostos na literatura para a implementação de Redes Neurais com Função de Base Radial (RNFBR), no entanto, há na literatura poucos trabalhos que abordam as aplicações práticas desses circuitos, em especial os que se utilizam das funções gaussianas como base destas redes.

Sabendo que a implementação física do circuito integrado (CI) traz grandes dificuldades em função de limitações e imperfeições inerentes aos processos de fabricação, seria interessante realizar o treinamento desta rede neural de forma inteligente, para minimizar os erros de processos excluindo a calibração manual.

As superações dessas dificuldades podem ser eliminadas através da programabilidade da rede com um circuito que garanta de forma satisfatória o controle dessa programação.

1.2) Objetivo

Fazer o treinamento de uma rede neural com um *hardware*, através do Algoritmo Genético de forma eficiente e robusta, verificando suas aplicações em problemas práticos.

1.3) Organização do texto

A dissertação foi dividida em 6 capítulos, além dos apêndices e referências.

No capítulo 2 é apresentada a idéia geral da rede neural de base radial, com sua proposta de funcionamento em dois blocos: bloco gerador de função gaussiana e bloco controlador de parâmetros.

No capítulo 3 são apresentados os conceitos teóricos e fundamentais sobre algoritmo evolutivo que neste projeto foi o Algoritmo Genético.

No capítulo 4 é apresentado a parte experimental do trabalho, com discussão do treinamento da rede e os pré-testes realizados com o circuito integrado utilizado para gerar as gaussianas e materiais de laboratórios utilizados no projeto.

No capítulo 5 são apresentados os resultados experimentais obtidos e as aplicações práticas.

No capítulo 6 são apresentadas as conclusões e propostas de trabalhos futuros.

Capítulo 2
REDE NEURAL DE BASE RADIAL

2.1) A Idéia

A rede neural de base radial foi escolhida em função da necessidade de superar os problemas decorrentes do processo de implementação, teve como função de base, a função gaussiana, que é obtida através da característica tangente hiperbólica do par diferencial bipolar que permite sua caracterização através do controle dos seus parâmetros que são o centro, a largura e amplitude [1]. A seleção desses parâmetros, através de blocos adjacentes baseados em fontes de correntes programáveis será controlada computacionalmente que tornará possível o aprendizado adaptativo da rede neural idealizada.

2.2) A Proposta

De forma geral, uma função qualquer y(x) pode ser representada pela soma ponderada de um número finito de funções de base f(.), conforme apresentado na equação 1.

$$y(x) = \sum_{i=1}^{M} w_i \times f(x, p_i) + w_0 \times B \qquad (1)$$

sendo:

M o número de funções de base;

w_i o peso referente à i-ésima função de base;

w_0 o peso referente ao *bias;*

B o *bias* (em geral, B=1);

p_i o vetor de parâmetros da i-ésima função de base que inclui fatores de forma e deslocamento.

Os parâmetros M, w_i, w_0 e p_i podem ser ajustados durante o processo de aprendizado da rede que utilizará essa função de base.

A equação 1, pode aproximar qualquer função arbitrária com um grau desejável de precisão, desde que se disponha de um número suficientemente grande de unidades intermediárias (camada oculta), e que a função de base $f(x,p_i)$ seja absolutamente integrável ou uma função monotônica limitada [1].

Na figura 1, apresenta-se o diagrama esquemático de uma Rede Neural com Função de Base Radial (RNFBR), baseada na equação 1.

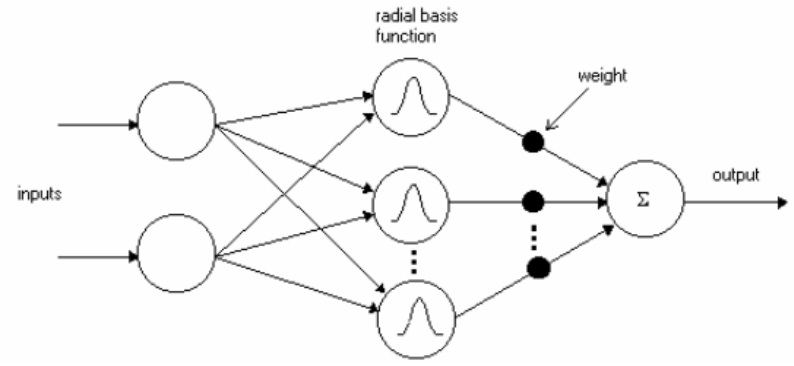

Figura 1 – Diagrama esquemático de uma RNFBR

As funções radiais formam uma classe especial de funções que se enquadram nos requisitos citados e têm como característica básica a apresentação de uma resposta monotônica crescente ou decrescente, a partir de um centro[1].

A função gaussiana faz parte dessa classe e pode ser submetida ao processo de aprendizagem que é proposto em quatro etapas:

1) Determinação do número de unidades de camada intermediária;
2) Localização dos centros;
3) Determinação da largura das funções;
4) Ajuste dos pesos de ligação entre as camadas intermediárias e de saídas.

Essas etapas de aprendizado podem ser realizadas através de várias metodologias e esse processo é destacado pelo fato de ser um processo fácil e rápido. Mas, essas características podem ser influenciadas por vários fatores relacionados a seguir:

- Função de base radial utilizada;
- Dimensão do vetor de entrada;
- Grau de liberdade permitido no ajuste dos parâmetros da rede (centro, largura e amplitude).

Desses fatores, os dois últimos são realmente importantes e indicam a viabilidade ou não, de utilizar uma rede neural de base radial na solução de um determinado problema.

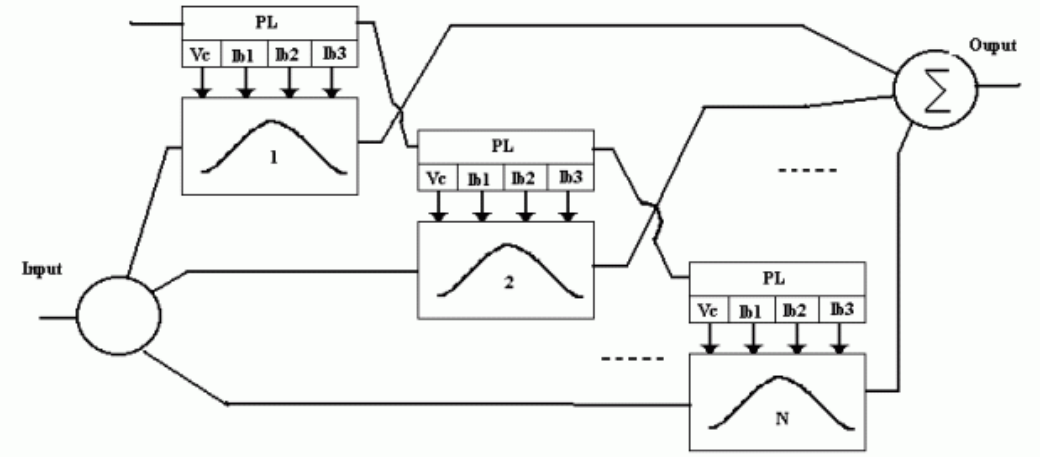

Figura 2 – Diagrama básico da rede neural idealizada

Na figura 2, apresenta-se um diagrama básico da RNFBR idealizada.

- A primeira camada (*input*) é linear e apenas distribui o sinal de entrada;
- A segunda camada (*1, 2,* ...N) é não linear e realiza a função de base radial;
- A terceira camada (*output*) realiza a combinação linear das saídas gaussianas.

Como se pode observar, a segunda camada pode conter tantos blocos quantos forem necessários para que o número de amostras satisfaça a necessidade das aplicações. Cada bloco inserido na segunda camada é constituído por dois grandes sub-blocos.

O sub-bloco principal realiza a função gaussiana e o outro sub-bloco realiza o controle dos parâmetros dessa função. Esses sub-blocos serão analisados separadamente

como dois blocos independentes apresentados nos sub-itens 2.3 e 2.4, respectivamente considerando a relação de interdependência entre os mesmos.

2.3) Bloco Gerador de Função Gaussiana

Neste sub-item serão apresentadas as três primeiras etapas de projeto do principal bloco do sistema, responsável pela geração da função de base da rede neural.

2.3.1) Caracterização

O circuito proposto se baseia na curva característica de transcondutância de um par diferencial bipolar. O par diferencial apresentado na figura 3 é um dos circuitos mais utilizados na implementação de redes neurais e gera como saída, a função diferença entre as duas tensões de entrada conforme descrito na figura 3.

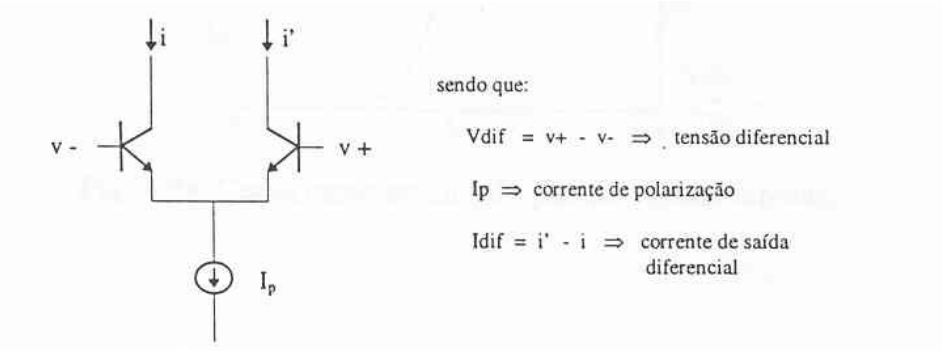

Figura 3 - Par diferencial

A equação que caracteriza o par diferencial é dada pela equação 2:

$$Idif = \alpha_F \times I_p \times \tanh\left(\frac{Vdif}{2 \times Vt}\right) \qquad (2)$$

sendo:

α_F = relação entre as correntes de coletor e emissor do transistor bipolar;
V_t = equivalente térmico de tensão.

A curva característica de saída do par diferencial é mostrada na figura 4.

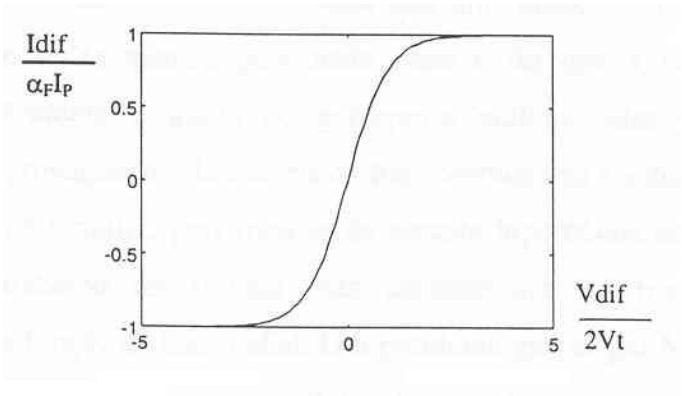

Figura 4 - Curva característica do par diferencial

A equação de transcondutância do par diferencial é dada pela equação 3:

$$g = \frac{\partial Idif}{\partial Vdif} = \frac{\alpha_F \times Ip}{2 \times Vt} \times \sec h^2 \times \left(\frac{Vdif}{2 \times Vt} \right) \tag{3}$$

sendo:

g = transcondutância do par diferencial.

A curva característica de transcondutância do par diferencial é apresentada na figura 5.

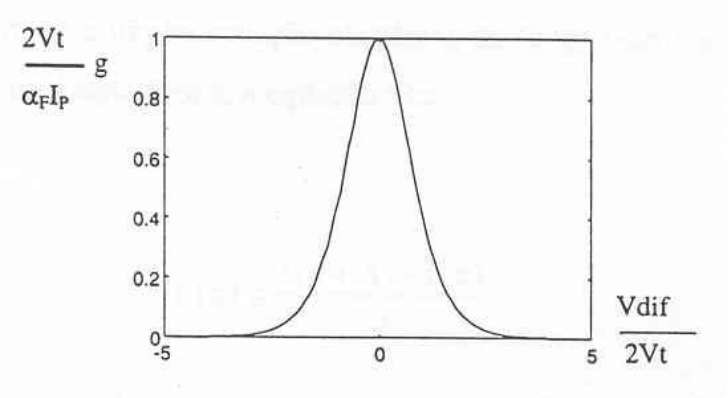

Figura 5 - Curva de característica de transcondutância do par diferencial.

Essa curva é uma secante hiperbólica quadrática derivada da função tangente hiperbólica relativa à curva característica do par diferencial. Essa derivada pode ser aproximada por uma diferença conforme descrito na equação 4.

A fórmula da derivada de uma determinada função f(x) é dada por:

$$f'(x) = \lim_{Vdelta \to 0} \frac{f(x + V_{delta}) - f(x)}{V_{delta}} \quad (4)$$

Considerando *delta* suficientemente pequeno, consegue-se uma boa aproximação da derivada. Assim, com dois pares diferencias e uma pequena tensão de *offset* (V_{delta}) aplicada na entrada de um deles, garante uma implementação eletrônica da função derivada bem próxima da desejada conforme a equação 5.

$$f'(x) \times V_{delta} \equiv f(x + V_{delta}) - f(x) \quad (5)$$

Então, conforme as equações 4 e 5, para gerar a função gaussiana, o bloco utiliza a característica de um par diferencial cuja resposta é uma tangente hiperbólica que quando diferenciada, gera uma secante hiperbólica quadrática que se aproxima muito da função gaussiana apresentada na equação 6.

$$f(x, w, b, c) = w \times \exp[-b(x - c)^2] \quad (6)$$

sendo:

c = centro;

b = largura;

w = amplitude.

A implementação do circuito foi feita com Amplificadores Operacionais de Transcondutância (OTA), o que garantiu maior facilidade de implementação e casamento entre componentes, sem perder a funcionalidade do par diferencial.

O OTA consiste basicamente em um par diferencial na entrada e alguns espelhos de corrente, que realizam a subtração das correntes de coletor dos transistores que compõem o par diferencial, de acordo com a figura 6.

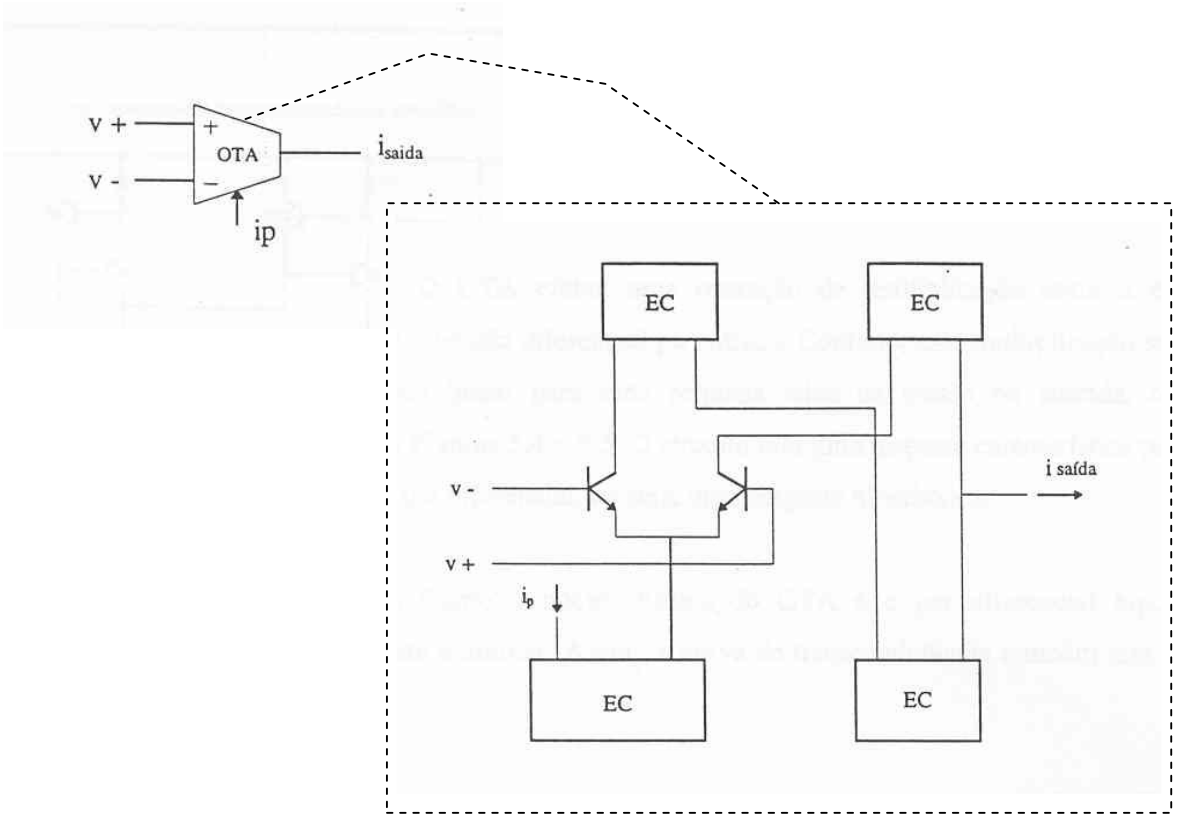

Figura 6 - Esquema de um OTA.

O comportamento do OTA é descrito pela equação 7.

$$I_{saída} = g \times I_P \times V_{ent} \qquad (7)$$

sendo:

g = transcondutância do OTA;

I_p = corrente de polarização;

V_{ent} = tensão diferencial na entrada.

O OTA efetua uma multiplicação entre a corrente de polarização e a tensão diferencial na entrada e apresenta como resposta característica uma curva praticamente idêntica a do par diferencial. Essa mesma similaridade pode ser considerada para a curva de transcondutância.

A programabilidade [1] dos parâmetros dessa aproximação de função é obtida através da utilização desses OTA's na implementação deste bloco. Assim, um esquema completo simplificado do sistema todo é mostrado na figura 7.

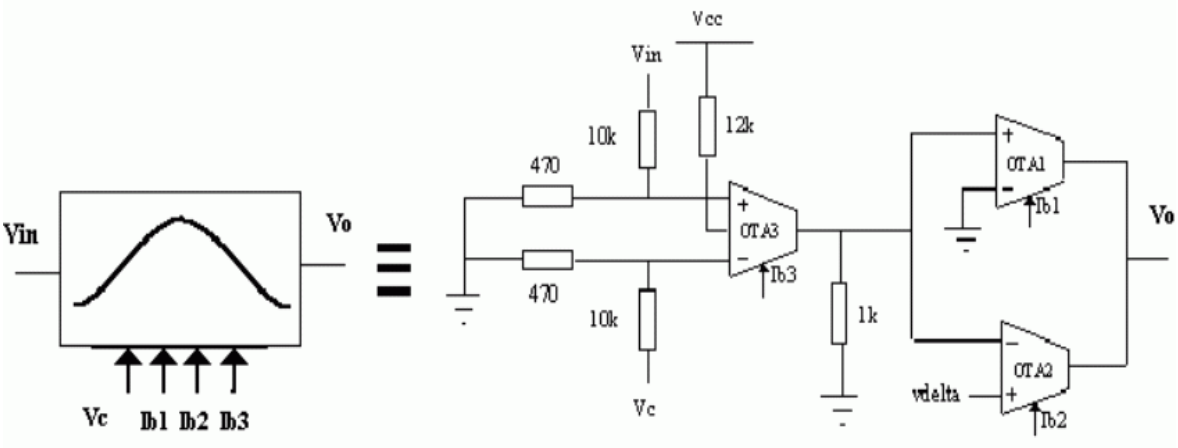

Figura 7 – Arquitetura proposta para o circuito gerador da função gaussiana.

O OTA, além de preservar a característica tangente hiperbólica da resposta do amplificador diferencial bipolar, permite o controle do ganho de transcondutância linearizado [1] através de corrente induzida.

O circuito proposto é mostrado na figura 7, onde o primeiro amplificador de transcondutância (OTA_3) possui uma rede de linearização [3] na entrada e executa a função de multiplicador permitindo que se varie o centro e a largura da função através das entradas de tensão Vc e de corrente I_{b3} respectivamente.

Essa rede de linearização é normalmente utilizada nos OTA's para adequá-los a aplicações em circuitos lineares normalmente controlados por corrente e que exigem uma faixa linear mais abrangente na entrada.

A saída linearizada [3] do OTA_3 pode ser representada conforme a equação 8 e o esquemático deste circuito de linearização é apresentado na figura 8.

$$Iout = g_m \times (V_{(+)} - V_{(-)}) \qquad (8)$$

Os outros dois amplificadores, OTA_1 e OTA_2, realizam uma função secante hiperbólica quadrática fixa em relação ao centro e a largura e possibilitam o controle da amplitude dessa função através de I_{b1} e I_{b2}. Essa função é a derivada da tangente

hiperbólica realizada no OTA_3, obtida através da aplicação de uma pequena tensão fixa de *offset* em V_{delta}.

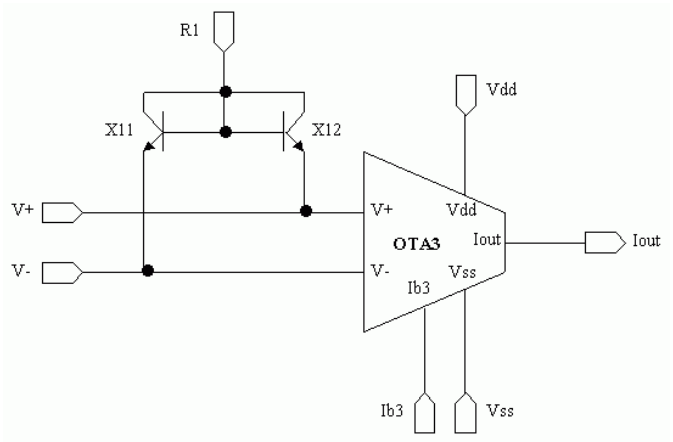

Figura 8 – O OTA de entrada com os diodos de linearização.

Na figura 8, os transistores bipolares X11 e X12 são diodos conectados para a característica de linearização do OTA_3[2].

Pela figura 7, nota-se que o circuito possui quatro entradas necessárias ao controle da função gerada, sendo uma delas controladas por tensão e as outras por corrente, conforme descriminado a seguir:

Vc = Tensão induzida para o controle do centro da gaussiana;

I_{b3} = Corrente induzida para o controle da largura da gaussiana;

I_{b1} e I_{b2} = Correntes induzidas para o controle da amplitude da gaussiana.

2.4) Bloco Controlador de Parâmetros

Este sub-item apresenta o funcionamento do sistema que controla os parâmetros da função gaussiana.

2.4.1) Introdução

A figura 9 apresenta o bloco que controla de maneira independente as quatros entradas vistas anteriormente. São três fontes de corrente, uma fonte de tensão e um registrador de deslocamento para facilitar a programabilidade do gerador das gaussianas.

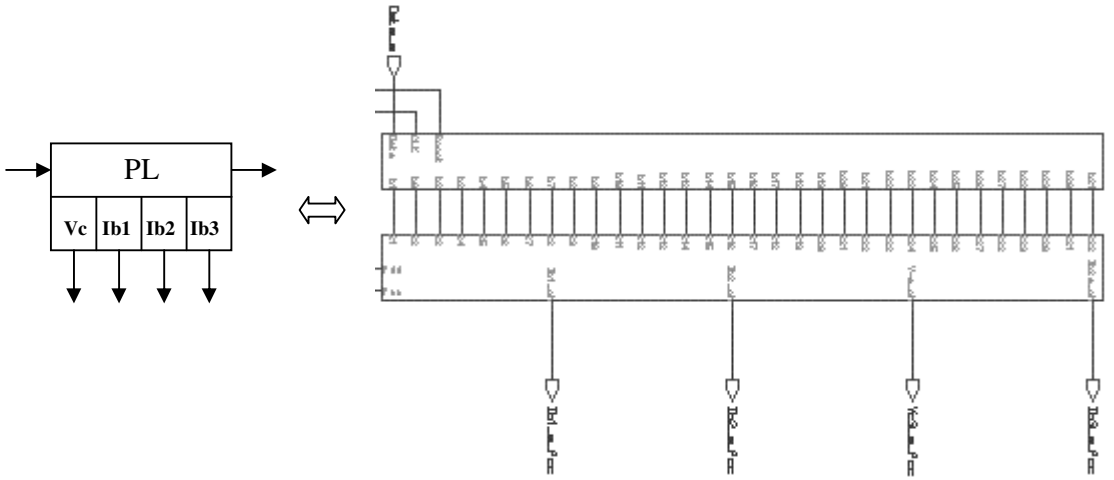

Figura 9 – Arquitetura proposta para o circuito controlador de parâmetros

O circuito possui uma entrada de dados serial para receber a palavra de controle. Essa palavra de dado de controle é enviada para a porta paralela do PC. Um breve estudo da porta paralela é apresentado no Apêndice A.

Através do registrador de deslocamento os dados em formato binário são distribuídos para seus respectivos circuitos fontes, que são chaveados conforme as informações obtidas desses dados de controle.

Um conjunto de 8 bits é fornecido para cada fonte, totalizando 32 bits para o circuito completo. Para cada fonte existe um multiplicador de corrente que recebe essa palavra digital de 8 bits [30] que determina o chaveamento correto do circuito.

2.4.2) Funcionamento

Para facilitar a programabilidade das quatros fontes e um registrador de deslocamento foram utilizados multiplicadores de corrente, baseados em espelhos de corrente para fornecer os sinais de controle ao gerador. A figura 10 mostra o circuito com mais detalhe.

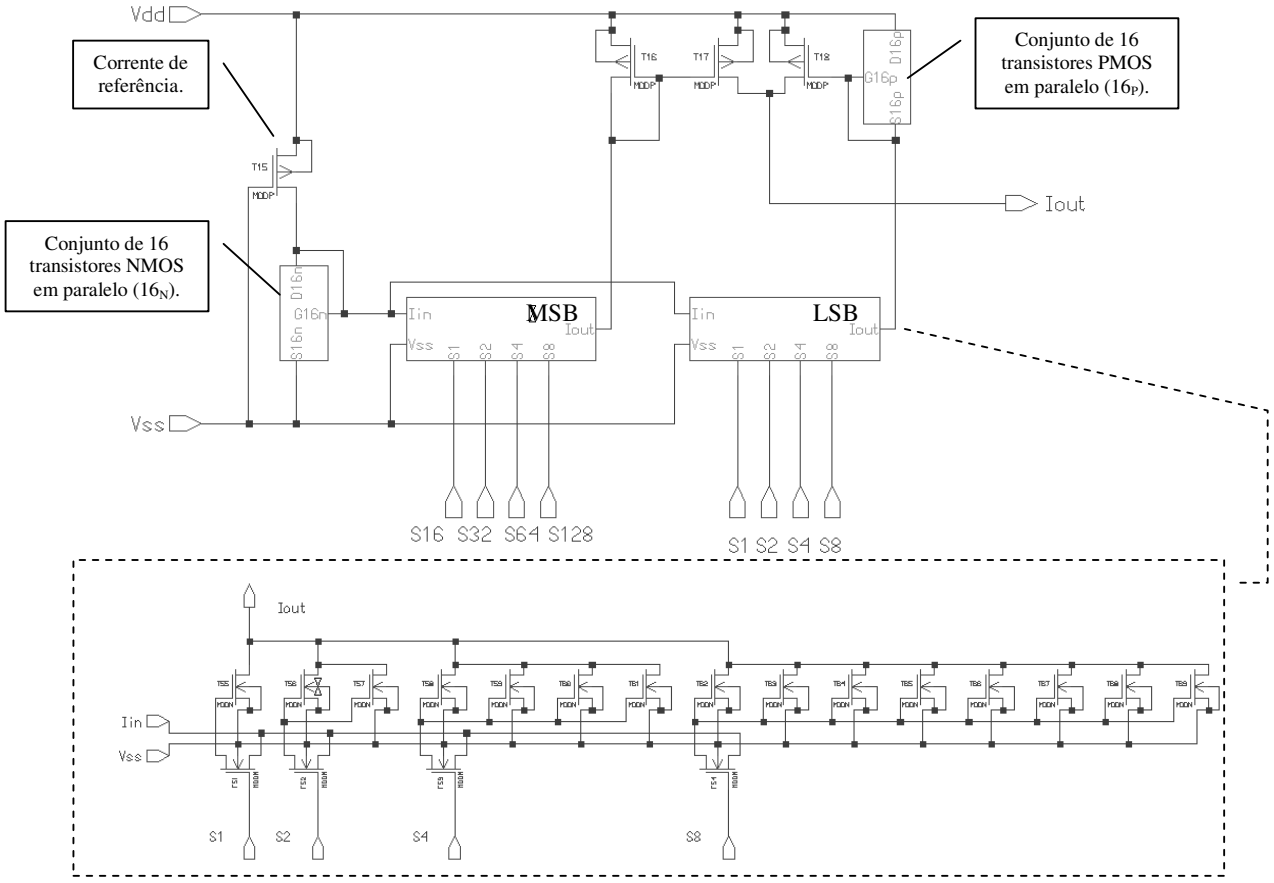

Figura 10 – Circuito detalhado do multiplicador de corrente.

Verificando o funcionamento do circuito da figura 10, nota-se que uma corrente de referência é produzida pelo transistor T_{15} e divide por 16 através do conjunto 16_N e

dispõe em um espelho de corrente no qual, dois conjuntos de 1, 2, 4 e 8 transistores em paralelo podem ser habilitados através de chaves (S_1 – S_{128}), que permitirá ou não a passagem de corrente de acordo com a palavra digital inserida pelo registrador de deslocamento.

As correntes geradas pelo chaveamento dos transistores são inseridas separadamente em dois outros espelhos de corrente. Em um dos espelhos a corrente é dividida por 16 novamente através do outro conjunto 16_P para que os 4 *bits* de controle desse conjunto sejam os menos significativos (*Last Significant Bit - LSB*).

No outro espelho a corrente não é dividida para caracterizá-los como os *bits* mais significativos (*Most Significant Bit - MSB*).

Somando as correntes desses dois espelhos que é gerada pelos conjuntos dos *bits* mais significativos e menos significativos, gera a corrente final (I_{out}), dada pela equação 9.

$$I_{OUT} = I_{IN} \times \left[2^{-1}.b_0 + 2^{-2}.b_1 + 2^{-3}.b_2 + 2^{-4}.b_3 + 2^{-5}.b_4 + 2^{-6}.b_5 + 2^{-7}.b_6 + 2^{-8}.b_7\right] \quad (9)$$

Um conversor corrente-tensão é utilizado, pois um dos controles é realizado em função de uma tensão.

O conversor é colocado na saída de um dos multiplicadores de corrente, fazendo com que a fonte de tensão funcione com a mesma lógica das fontes de corrente, e este é apresentado na figura 11. Este conversor corrente-tensão fornece a tensão de controle Vc e tem a característica de variar linearmente a tensão de saída em relação à corrente de entrada.

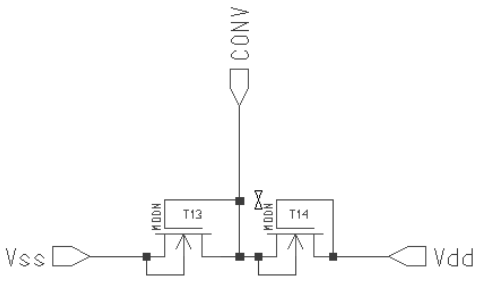

Figura 11: Conversor Corrente-Tensão.

O conversor corrente-tensão possui dois transistores do tipo NMOS que funcionam como resistores. Assim, a corrente que passa pelo conversor, produz uma tensão proporcional (V_{out}) no terminal CONV.

A equação 10 representa o comportamento resistivo linear do conversor da figura 11.

$$\frac{V_{out}}{I_{in}} = \frac{1}{2K\left(V_2 - 2V_t\right)} \tag{10}$$

Finalmente, as quatro fontes de corrente necessárias ao circuito de controle do gerador de gaussianas são apresentadas na figura 12.

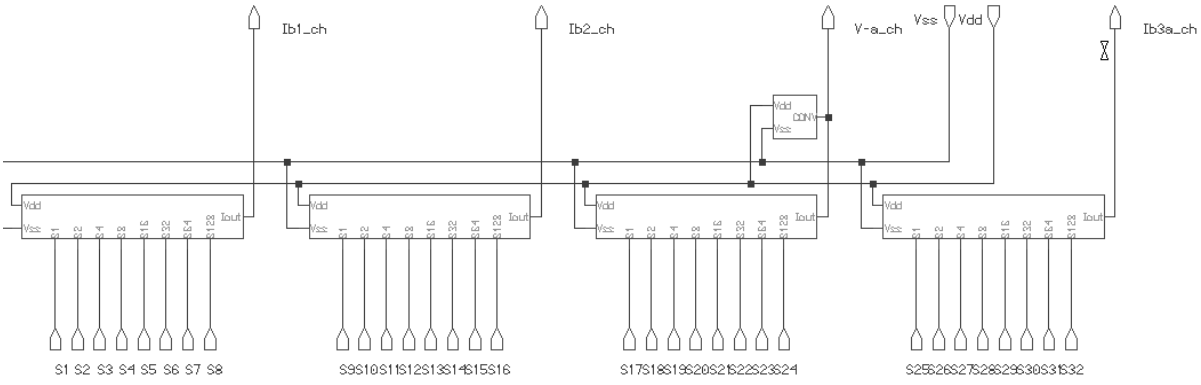

Figura 12: Circuito completo das quatro fontes
de controle para geração de uma gaussiana.

Como apresentado na figura 12, quatro circuitos idênticos foram inseridos no CI, de forma a utilizar quatro funções de base para o treinamento da rede neural em questão. Um dos circuitos para a geração das gaussianas foi descartado por ter sido queimado na fase de testes do CI e, portanto, utilizaram-se apenas três gaussianas, necessárias para o treinamento da RNFBR.

De maneira a facilitar a programabilidade do circuito da figura 12, foi implementado um registrador de deslocamento apresentado na figura 13, de forma que as quatro palavras binárias de 8 bits fossem programadas serialmente, transformado em

uma palavra de 32 bits. Com isso, o CI exigia uma única entrada para a distribuição das 32 entradas das fontes de controle.

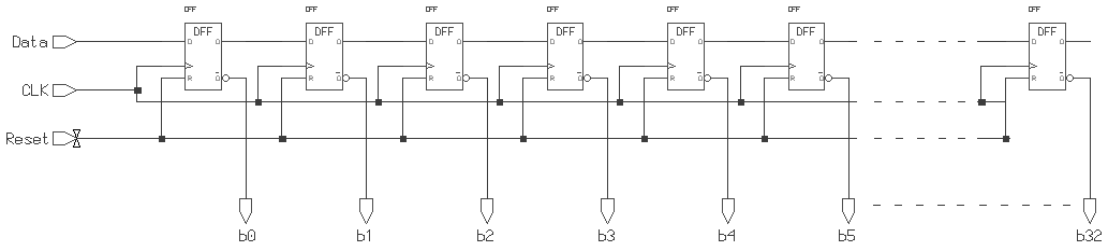

Figura 13: Registrador de deslocamento.

O registrador apresentado na figura 13 tem como objetivo facilitar a programabilidade das fontes de correntes.

Todo treinamento da rede neural e envios de dados ao CI foram realizados com a programação em linguagem C (Turbo C).

Capítulo 3

ALGORÍTMO EVOLUCIONÁRIO

3.1) Introdução

No final da década de 50, alguns pesquisadores buscaram na natureza inspiração para novas técnicas de busca de soluções. O motivo para a atenção ter se voltado para a natureza deve-se ao fato desta conseguir resolver, de forma satisfatória, problemas altamente complexos, como é o caso da sobrevivência das espécies. Aliado a este fato, é possível explicar a grande maioria dos seres vivos através de poucos processos de natureza estática (cruzamento, mutação e seleção), agindo sobre uma população de uma espécie [22-23].

A tentativa de imitação do cérebro humano na expectativa de um comportamento emergente deu origem as Redes Neurais Artificiais (RNA). Já a tentativa de imitar a evolução dos seres vivos na natureza originou a Computação Evolucionária (CE).

A Computação Evolucionária é o nome genético, dado a métodos computacionais, inspirados na teoria da evolução. Os algoritmos usados, em computação evolucionária, são conhecidos como Algoritmos Evolucionários (AE).

Atualmente, os AE mais conhecidos são: Algoritmos Genéticos (AG), Programação Evolucionária (PE) e Estratégias Evolucionárias (EE). Todos compartilham de uma base conceitual comum, que consiste na simulação da evolução de estruturas individuais, via processos de seleção e os operadores de busca, referidos como Operadores Genéticos (OG), tais como, mutação e recombinação. O processo depende do "*fitness*" (aptidão). Atingido pelas estruturas individuais, frente a um ambiente. A seleção é focalizada nos indivíduos com um alto grau de aptidão, explorando então, a informação da aptidão disponível. A recombinação e a mutação perturbam estes indivíduos, fornecendo heurística geral para a exploração.

O AG foi proposto inicialmente por John H. Holland em 1975 no trabalho intitulado "*Adaptation in Natural and Artificial Systems*" [24]. Holland inspirou-se no mecanismo de evolução das espécies, tendo como base os trabalhos de Darwin sobre a

origem das espécies [25] e na genética natural, devido principalmente a Mendel [26]. De acordo com a teoria Darwiniana de evolução das espécies, uma população sujeita a um ambiente qualquer, sofrerá influência desse, de tal forma que os aptos terão maior probabilidade de sobreviver a tal ambiente.

Já os trabalhos de Mendell mostram como o material genético dos pais pode ser passado para os descendentes. Desta forma, a cada geração haverá uma população mais adaptada ao ambiente em questão [24][27].

Os AG podem ser enquadrados, em grande parte dos problemas científicos a serem formulados, como problemas de busca e otimização.

Os AG fazem parte da classe correspondente aos métodos probabilísticos de busca e otimização, apesar de não serem aleatórios. Os AG usam o conceito de probabilidade, mas não são simples buscas aleatórias. Pelo contrário, os AG tentam direcionar a busca para regiões onde é provável que os pontos ótimos estejam.

Além disso, em relação às técnicas de busca convencionais, os AG diferem nos seguintes pontos:

- A busca da melhor solução para o problema é feita sobre uma população de pontos, e não sobre um único ponto, reduzindo sensivelmente o risco da solução recair sobre um máximo (ou mínimo) local;
- Os AG realizam uma busca cega. A única exigência é o conhecimento do valor da função aptidão (ou *fitness*) de cada indivíduo. Não há necessidade de qualquer outra informação, ou heurística, dependente do problema.
- Os AG usam operadores estocásticos e não regras determinísticas para guiar uma busca altamente exploratória e estruturada, onde informações acumuladas nas iterações (gerações) anteriores são usadas para direcionar essa busca.

Apesar de sua simplicidade, os resultados obtidos com a aplicação do método, segundo Goldberg [27], permitem concluir que os AG são um método de busca robusto, eficiente e eficaz em uma grande variedade de problemas.

3.2) Conceitos Fundamentais

Em termos biológicos, um indivíduo é formado por um conjunto de cromossomos. No entanto, pode-se fazer uma analogia, neste contexto, entre indivíduo e cromossomo, tendo em vista que um indivíduo pode ser formado por apenas um cromossomo, o que é comum em AG. Por isso, os dois termos são utilizados indistintamente, neste contexto.

Cada indivíduo é definido por um *string*, usando-se um alfabeto finito, de modo que cada *string* represente um conjunto de valores para o conjunto de parâmetros do problema. Um exemplo de alfabeto é o conjunto {0,1}, ou o conjunto de números inteiros. Deste modo, cada posição do *string* representa um gene.

O cromossomo é composto de genes sendo que cada gene possui um local fixo no cromossomo, local este denominado de "*locus*". Cada gene pode assumir um certo valor, pertencente a um certo conjunto de valores, os quais, são denominados de "alelo"[28]. Em termos de AG, o gene é denominado de "*bit*" e o "*locus*", de posição do "*bit*" no indivíduo. Já o termo alelo, refere-se ao conjunto de valores possíveis de serem atribuídos a um determinado "*bit*".

Normalmente, os AG trabalham com um conjunto de indivíduos (população) , no qual, cada elemento é candidato a ser a solução desejada. Cada indivíduo é codificado em uma cadeia de bits, denominada de cromossomo. A cadeia de cromossomos, representada por números binários é de comprimento fixo. A função a ser otimizada é o ambiente, no qual a população inicial vai ser posta. Espera-se que, através dos mecanismos de evolução das espécies e a genética natural, somente os mais aptos se reproduzam e, também, que cada nova geração esteja mais apta ao ambiente (função a ser otimizada).

O grau de aptidão de cada indivíduo é obtido pela avaliação de tal indivíduo, através da função a ser otimizada. Se o objetivo for maximizar, a aptidão é diretamente proporcional ao valor da função. Caso o objetivo seja a minimização da função, a aptidão será inversamente proporcional ao valor da função[29].

Assim, quando já se tem realizado o teste de todos os indivíduos da população, na função a ser otimizada, obtém-se a aptidão para cada um, ou seja, o seu grau de aptidão. A próxima geração será uma evolução da anterior e, para que isso ocorra, os mais aptos, os de melhor aptidão, deverão possuir maior probabilidade de serem

selecionados para dar origem à nova geração. Com isso, se o processo for bem conduzido, espera-se que a nova geração seja, em média, melhor do que a que lhe deu origem.

A seleção dos indivíduos da geração anterior, que vão participar da formação da nova geração pode ser feita através de sorteio por torneio como foi realizado no projeto em questão, onde se sorteiam três números aleatórios indicando cada indivíduo da população inicial e aquele que tiver o melhor *fitness* é o escolhido para fazer parte da nova geração. Feito isso, a nova geração terá uma aptidão melhor que a geração anterior. O sorteio por torneio é feito até completar a nova geração.

Entre tais mecanismos, os mais comumente empregados em AG, são: recombinação (*crossover*) e mutação. Estes operadores serão descritos com mais detalhes, nos próximos sub-itens.

Um outro ponto relevante a ser mencionado, diz respeito aos parâmetros do AG, ou seja, os valores que influenciam o desempenho do AG. Seguindo a relação proposta por S. Austin [12], estes parâmetros são: tamanho da população, taxa de operadores, intervalo de geração e seleção de estratégia.

O tamanho da população de cromossomos afeta o desempenho global dos AG. Uma população pequena é insuficiente para cobrir o espaço de busca do problema. Uma população grande é mais representativa do domínio, além de evitar a convergência prematura para soluções locais, em vez de soluções globais.

As taxas de operadores medem a freqüência com que cada tipo de OG é utilizado. Representam, também, a influência que cada tipo de OG exerce sobre a população de cromossomos. Além do que, se a taxa de operadores de *crossover* for muito alta, alguns cromossomos de bom desempenho podem ser removidos mais rapidamente do que a seleção possa desenvolvê-los. Se a taxa de *crossover* for muito baixa, a busca pode estagnar. Entretanto, se a taxa dos operadores de mutação for baixa, evita-se que uma dada posição estabilize-se em um único valor. Uma taxa de mutação alta resulta essencialmente numa busca aleatória.

As estratégias de seleção correspondem aos critérios utilizados para a escolha de cromossomos durante a reprodução. Um exemplo de estratégia de reprodução, baseado em S. Austin [12], é a seleção pura, onde os cromossomos são reproduzidos em função da sua aptidão.

3.3) Operadores Genéticos

Indivíduos novos são criados usando-se dois operadores de recombinações genéticas, conhecidas como *crossover* e mutação.

O *crossover* se dá pela aproximação dos cromossomos dos dois indivíduos (pais), que trocam entre si partes de seus cromossomos. Isso resulta em dois cromossomos diferentes que, porém, ainda guardam influências dos pais [7]. Há várias formas possíveis de se fazer o cruzamento [13][14][15][16]. O operador *crossover*, mais simples, é o chamado *crossover* de um ponto, onde, primeiro um local de cruzamento é escolhido com probabilidade uniforme sobre o comprimento do cromossomo, sendo, então, os *strings* correspondentes permutados, como é mostrado na Figura 14. Há, ainda, muitas outras técnicas de *crossover*, como é o caso do *crossover* de dois pontos (*Two-Point*), e dos tipos uniformes [16][17]. Contudo, não há consenso sobre qual é a melhor técnica a ser usada.

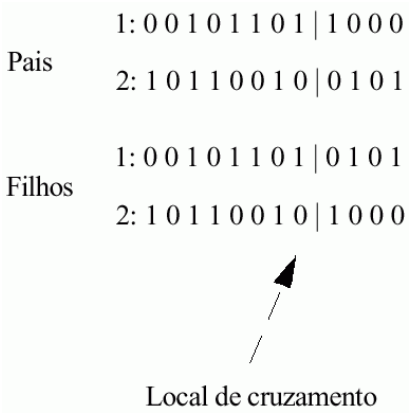

Figura 14 - Operador *crossover* de um ponto.

A mutação consiste em perturbações na cadeia dos cromossomos dando origem a uma nova cadeia, que guardará pouca ou nenhuma informação da cadeia mãe. Na realidade, mutação é a denominação dada a vários mecanismos de alteração genética, os quais têm em comum o fato de fazerem o novo cromossomo apresentar pouca informação dos pais [18]. Esta alteração ocorre de forma que cada gene em cada cromossomo é um candidato à mutação, enquanto que a seleção é determinada pela probabilidade de mutação. Esta probabilidade é mantida, usualmente, em um valor

baixo, para evitar-se a perda de um número grande de cromossomos bons [15]. O operador de mutação pode ser implementado de várias maneiras. A codificação binária de *string* é o modo mais fácil para executá-la.

A tarefa da mutação em AG tem sido a de restituir a perda ou material genético inexplorado na população, com o objetivo de prevenir a convergência prematura do AG para soluções sub-ótimas [16].

O mecanismo de alteração genética para este trabalho foi a troca simples que consiste de um erro de cópia de um ou mais genes da cadeia. Se um gene for considerado como sendo um bit com valor lógico 1, a ocorrência de troca simples levaria este bit (gene) para nível lógico 0 e vice-versa (Figura 15).

Na maioria dos trabalhos com AG, este também usa o termo mutação como sinônimo de troca simples [13] [18] [19] [24].

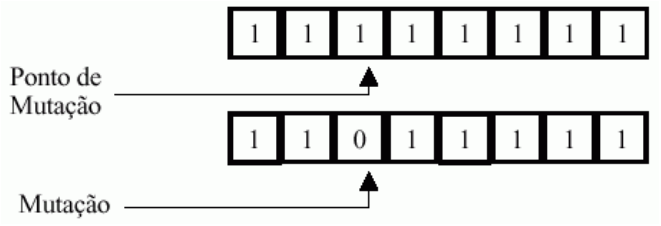

Figura 15 - Exemplo de mutação (troca simples).

Por último, após a seleção e a aplicação dos OG, tem-se uma nova geração, a qual deve ser avaliada, visando comparar o seu grau de aptidão (*fitness*) em relação a geração anterior. Caso tal geração não esteja apta o suficiente, deve-se repetir o processo de seleção e reprodução, até que o grau de aptidão seja aceitável [13][18].

3.4) Algoritmo Genético Simples

O trabalho original de Holland [16] propõe os seguintes passos principais para um algoritmo genético simples:

- Geração da população inicial;
- Validação dos elementos da população e análise de convergência;
- Seleção;
- Manipulação genética.

3.4.1) Geração da População Inicial

A população inicial pode ser obtida através da geração aleatória de indivíduos, obedecendo condições de contorno previamente estabelecidas pelo usuário. O usuário estabelece estas condições, tendo em vista o seu conhecimento prévio do problema a ser otimizado. Quanto mais restringentes forem as condições de contorno, mais rápida será a convergência, pois os valores gerados aleatoriamente estarão mais próximos da solução desejada[13].

O número de elementos, que comporá a população, ainda é motivo de estudos, mas existem várias heurísticas, ou seja, depende muito da experiência do usuário e do seu conhecimento prévio sobre a função a ser otimizada [6]. É claro que, quanto maior o número de elementos na população, maior é a probabilidade de convergência, tendo em vista que aumenta a probabilidade da solução desejada ser constatada entre os elementos da população. Em contrapartida, o tempo de processamento também aumenta. Já, no caso da população inicial ser muito pequena, ela terá o problema da perda de diversidade, isto é, o espaço de busca seria muito pequeno para ser avaliado. Desta forma, a solução obtida poderia não estar dentro do ótimo global. Conseqüentemente, a convergência seria prematura.

O número de elementos na população, a probabilidade de ocorrer cruzamento e a probabilidade de acontecer mutação, são denominados de parâmetros de controle dos AG [16].

3.4.2) Validação dos Elementos da População e Análise de Convergência

A validação é o processo de expor cada elemento da população a função objetivo e, ao final, ordená-los de acordo com a aptidão à esta função. Na convergência, analisa-se o desempenho da população para ver se o objetivo foi atingido. Isto pode ser feito através de vários fatores, tais como: valores máximo, mínimo e médio da função de aptidão. Também, é relativamente comum utilizar-se o desvio padrão dos valores da função de aptidão, como forma de análise da convergência [27]. Como o AG é regido por população, se na população inicial tiver um elemento que seja a resposta exata do problema, o AG ainda assim não finalizará o processo de busca da solução. A

finalização ou convergência só ocorrerá quando a aptidão média da população estiver suficientemente estável, ou seja, quando houver pouca variação da aptidão média da população atual em relação a anterior. Isto indica que a população se adaptou ao meio, isto é, os elementos da população levam a função ao valor otimizado/desejado [16].

Utiliza-se memorizar o indivíduo mais apto, independentemente deste fazer, ou não, parte da população atual. Assim, ao final, este será o resultado esperado [29].

Contudo, na utilização de AG pode ocorrer uma rápida convergência prematura para uma solução sub-ótima, porém não o esperado ótimo global. Este problema é denominado convergência prematura, podendo ocorrer devido à população reduzida ou a má distribuição da população inicial, em torno do ponto sub-ótimo. Ou seja, um indivíduo próximo de um ótimo local, possui um valor de aptidão superior aos demais indivíduos da população.

Conseqüentemente, o processo de seleção fará com que este indivíduo tenha grande chance de dominar a próxima geração e, assim sucessivamente, se não aparecerem outros indivíduos com melhores valores de aptidão [28][29].

Conforme pode ser visto, a convergência prematura pode ocorrer devido a uma má distribuição dos indivíduos no espaço de busca. Esta má distribuição, também recebe a denominação de perda da diversidade [27][29]. Segundo J. Tanomaru [29], o conceito de diversidade indica o grau em que as mais diversas regiões estão representadas no espaço de busca. Este problema pode ser amenizado através da escolha criteriosa do número de indivíduos na população, melhora da distribuição dos indivíduos da população inicial no espaço de busca e, também, impedindo a perda de diversidade nas primeiras gerações.

3.4.3) Seleção

A seleção tem por objetivo fazer com que somente os elementos mais aptos da geração anterior participem do processo que irá gerar a nova população. O processo de seleção tem início após a verificação do grau de aptidão (*fitness*) de cada elemento à função de custo e a verificação da não convergência dos valores[7].

3.4.4) Manipulação Genética

A etapa de manipulação genética consiste na aplicação de OG, isto é dos operadores *crossover* e/ou mutação, somente em alguns elementos que tiveram maior valor de aptidão, quando sorteados por meio do torneio. Ao término desta etapa terá sido gerada uma nova população, que deverá repetir os passos anteriores até que a aptidão da população seja aceitável.

Inicialmente é necessário estabelecer alguns pontos importantes, tais como: manter o tamanho da população fixo e garantir que uma parcela da nova população seja composta por elementos obtidos por seleção da população anterior, além do que, o complemento seja de elementos manipulados pelos OG.

Quanto aos OG, costuma-se executá-los em seqüência nos AG simples, isto é, inicialmente aplica-se o OG de *crossover* e, após, o de mutação[27].

A aplicação do *crossover* implica na composição de 10 casais a partir dos 20 elementos, sendo que alguns serão acasalados e outros não. Para tanto se gera dois números aleatórios: o primeiro, entre 0 e 1, indicará a probabilidade de ocorrer *crossover* e, o segundo, o local da realização do *crossover*. Caso o primeiro número gerado seja inferior ao definido pelo usuário, como probabilidade de *crossover*, realiza-se o *crossover* propriamente dito, caso contrário copia-se os pais para a nova geração [29]. O segundo número aleatório, gerado no caso da ocorrência de *crossover*, indicará a posição de corte do cromossomo para efetuar o *crossover*. Para tanto o número aleatório gerado deverá estar entre 1 e g - 1, onde g é o número de genes ou *bits* do cromossomo.

Para a aplicação do OG de mutação, há necessidade de gerar um número aleatório para cada indivíduo. Este número randômico é denominado de probabilidade de mutação e deverá ser comparado com a probabilidade de mutação estipulada pelo usuário chamada de taxa de mutação. Caso seja inferior a esta, executa-se a mutação, caso contrário repete-se o processo para o próximo indivíduo, até que todos os indivíduos tenham sido analisados [27][29].

A probabilidade de ocorrer mutação é sempre bem menor que a de ocorrer *crossover*. Segundo M. Sirivas e L.M. Patnaik [16], existe um compromisso entre os três parâmetros de controle do AG simples: tamanho da população, taxa de *crossover* e taxa de mutação. Muitos autores têm proposto valores para estes parâmetros visando

garantir uma boa performance do AG, porém, estes valores ainda fazem parte de várias heurísticas [8].

3.5) Aplicações de AG para Redes Neurais

AG podem ser aplicados às Redes Neurais Artificiais (RNA) em três modos:

- treinamento de RNA (ajuste dos pesos);
- otimização da topologia de RN (número de neurônios da camada intermediária);
- geração tanto da topologia como dos pesos das conexões.

A maioria dos trabalhos em que AG são utilizados tratam do problema de treinamento, ou seja, da geração dos pesos das conexões da rede [8][13][29]. Neste tipo de aplicação, dada uma estrutura para uma rede, um AG é utilizado para achar os pesos que resultam nos menores valores de erro, ao invés de algoritmos de treinamento convencionais, como o "*Backpropagation*" (retropropagação). Por exemplo, Muhlenbein [9] abordou a dinâmica da evolução combinada com aprendizado, dando os primeiros passos para a compreensão dos dois mecanismos agindo simultaneamente[9]. Prado[10] e Porto[11] exploraram o treinamento de RNA diretas através de AG.

3.6) Fluxograma do Treinamento da Rede Neural

Os fluxogramas utilizados na execução dos programas de treinamento da rede neural estão apresentadas nas figuras 16 e figura 17.

A figura 16, representa o fluxograma de treinamento de uma senóide ideal e a figura 17 representa o fluxograma para a linearização de saída para o VCO.

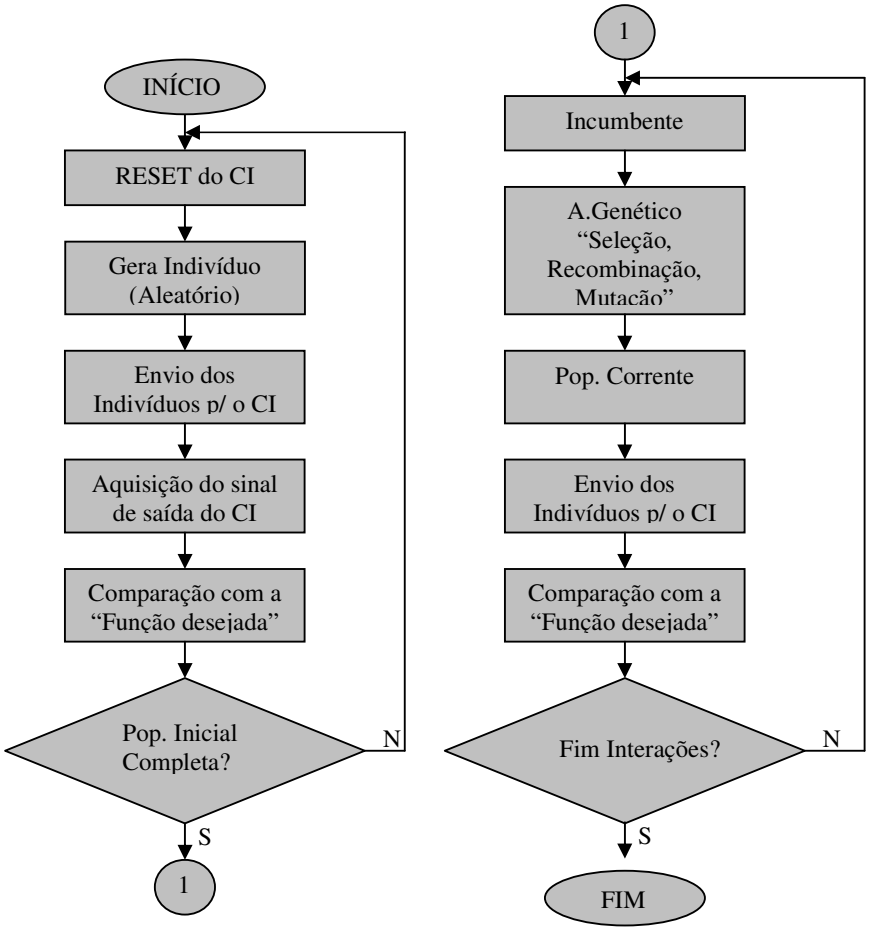

Figura – 16: Fluxograma de treinamento da rede neural para a senóide ideal.

Explicação de cada bloco do fluxograma da figura 16.

a) Reset do CI – É a etapa inicial do treinamento. Nesta parte o circuito integrado a ser treinado é inicializado.

b) Gera Indivíduo – Nesta parte do treinamento são gerados de forma aleatória através da programação em linguagem C, os *strings* de 128 bits que serão armazenados em uma matriz, que esta será a população inicial a ser analisada para o treinamento da rede neural.

c) Envio dos Indivíduos para o CI – Após a geração da população inicial, todos os *strings* desta matriz são enviados através da porta paralela do PC para o circuito integrado, onde cada *string* habilitará a fontes de correntes e tensão para a geração das gaussianas.

d) Comparação com a "função desejada" – Com a geração das gaussianas é obtido a saída da rede neural que foi comparada com a função senóide ideal para a aplicação do conversor de onda triangular-senoidal. Com a comparação gera-se o *fitness* de cada *string* através do erro quadrático médio, onde este faz a somatória da diferença ao quadráto das duas funções em questão.

e) Incumbente – Nesta etapa é capturado o *string* de melhor *fitness*, ou seja, aquele que possui o menor erro quadrático médio dentre toda a população que esta sendo analisada.

f) A. Genético "Seleção, Recombinação e Seleção" – Nesta etapa do programa é realizado a aplicação do algoritmo genético para alterar os pesos da rede. A seleção é feita através de torneios, onde escolhe-se de forma aleatória dois *strings*, fazendo a recombinação e a mutação. Isso é feito até completar a nova população corrente, apresentado no próximo bloco.

g) População corrente – Nesta etapa do programa, são gerados os novos *strings* após a aplicação do algoritmo genético.

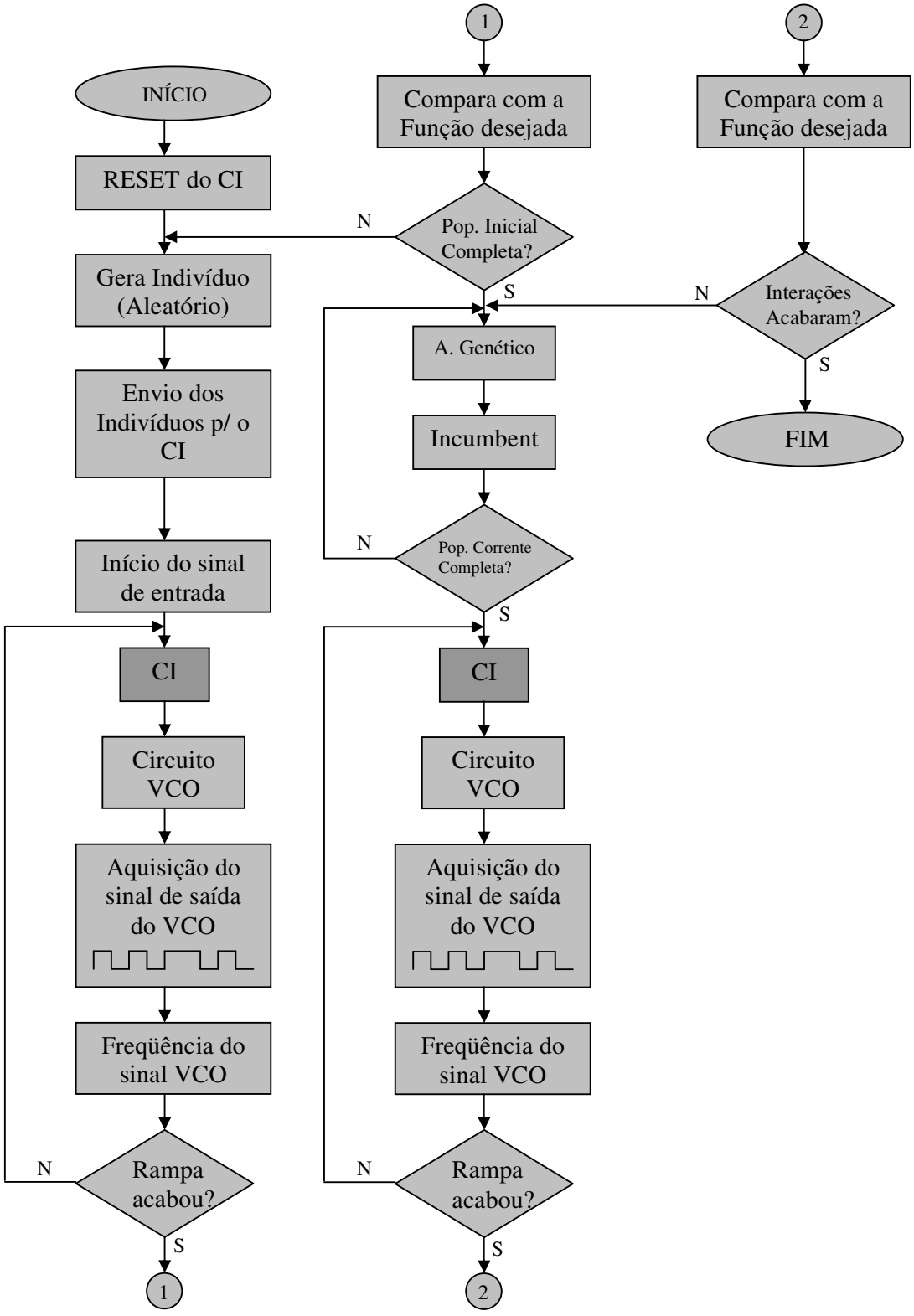

Figura – 17: Fluxograma de treinamento de rede neural para a linearização de saída do VCO.

Capítulo 4
PARTE EXPERIMENTAL

4.1) Introdução

O circuito proposto foi implementado com tecnologia BiCMOS 0.8μm da *AMS* e tem a capacidade de gerar funções gaussianas para fazer o treinamento da rede neural.

O circuito possui duas entradas e uma saída. Em uma das entradas é inseridos os pesos da rede neural, que são os *string's* de 128 *bits* gerado aleatoriamente. Na outra entrada é inserida uma tensão linear variável de 0 a 5V. Na saída somou-se as gaussianas, obtendo uma função relativa ao treinamento da rede neural.

Através da porta paralela do PC foi possível enviar as informações para o CI para realizar o treinamento da rede neural. Um breve estudo sobre a porta paralela foi realizado no Apêndice A.

Uma placa de aquisição de dados foi utilizada para captar o sinal de saída do circuito, para realizar a comparação da função desejada *(fitness)* e para isso, foi utilizado o erro quadrático médio como índice de *fitness* no treinamento da rede com o AG. No apêndice B, foi realizado um estudo sobre a placa de aquisição utilizada neste projeto.

Similar a aproximação da senóide ideal, montou-se o circuito da figura 18 utilizando componentes discretos e inseriu-se o sinal do CI que gera as gaussianas para realizar o treinamento da rede neural para a linearização da saída do VCO. Neste mesmo instante o sinal de entrada "rampa" que foi gerado pelo conversor D/A da placa de aquisição, foi inserido ao CI e para cada degrau da rampa foi obtido na saída do circuito completo uma respectiva freqüência. A figura 19, mostra a tensão de entrada (rampa) em função da freqüência para cada degrau da rampa. Com isso, o sinal de saída do VCO foi comparado com uma reta linear crescente, que através do erro quadrático médio fosse possível obter um índice de *fitness* para realizar o treinamento com o Algoritmo Genético. A figura 21 apresenta o resultado obtido do treinamento da rede, onde para cada tensão da rampa, captou-se sua freqüência.

O treinamento da rede e envio das informações para o CI foram feitas através da programação em Linguagem C.

Os programas utilizados para o treinamento da rede estão anexados no Apêndice C.

4.2) Linearização da saída de um VCO

VCO (Oscilador Controlado por Tensão) é um circuito que produz um sinal de saída oscilante (geralmente uma onda quadrada ou triangular) cuja freqüência é ajustada dentro de uma faixa, controlada pelo nível de uma tensão de entrada. Este tipo de circuito é usado para, entre outras coisas, modulação de sinais.

A linearização de um VCO tem o propósito de testar a performance da rede. A configuração implementada neste trabalho, utilizando o CI 555, é apresentado na figura 18. Por meio de uma tensão de controle (Vcon) aplicada no pino 5, a freqüência do oscilador pode ser variada numa determinada faixa[21]. Essa configuração foi escolhida de tal modo a evidenciar a não linearidade entre a tensão de controle e a freqüência na saída [20]. Os valores dos componentes utilizados são: Ra = 100kΩ, Rb = 10kΩ e C = 22nF.

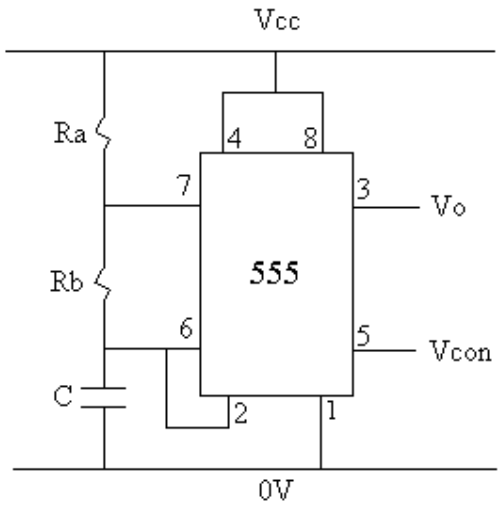

Figura 18 - Circuito VCO com temporizador 555

Na figura 19 é apresentada a característica de variação na freqüência de saída em função da tensão de controle Vcon.

A função característica do VCO é uma curva não linear decrescente como mostrado na figura 19.

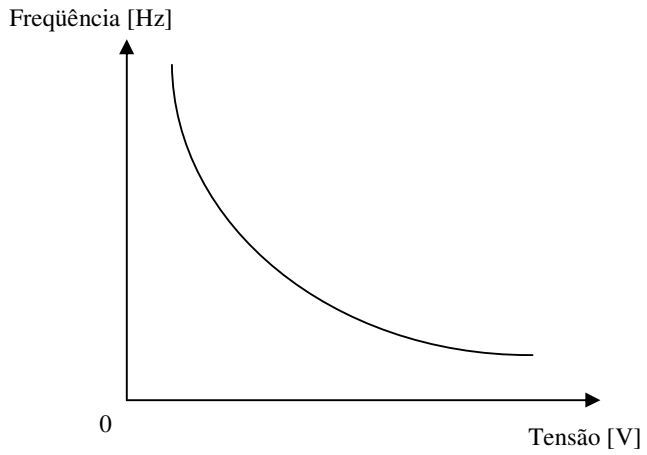

Figura 19: Gráfico da resposta do VCO em função de Vcon.

4.3) Pinos do Circuito Integrado para envio dos pesos da rede

Os principais pinos do CI para o envio das informações de dados foram:

- Pino 2: Recebe os dados binários (*string* de 128 *bits*), que são os pesos da rede neural. Estes *string's* são gerados de forma aleatória através da programação e enviados de forma serial sendo distribuído corretamente às fontes de corrente e tensão, com o auxílio de um registrador de deslocamento.

- Pino 4: É o RESET e quando em nível lógico alto, reseta o C.I.

- Pino 1 e 3: É o *clock* mestre e o *clock* respectivamente. Quando o *clock* envia os 32 *bits*, o *clock* mestre é acionado para enviar mais 32 *bits* até completar os 128 *bits*.

4.4) Instrumentos utilizado e local de projeto

Todo projeto foi realizado no laboratório de automação e microeletrônica na faculdade de engenharia de Ilha Solteira. Neste projeto foram utilizados um PC, placa de aquisição, um osciloscópio digital, uma fonte de alimentação simétrica, placas de *protobord* e um gerador de função digital.

4.5) Pré-testes realizados com o CI

Testes antecipados como o CI foram realizados para antever certos problemas que poderiam ser minimizados com o treinamento da rede neural.

Três problemas foram verificados no CI. Um deles foi a não funcionalidade de uma das gaussianas implementadas no CI, onde foram utilizados apenas três das quatro gaussianas existentes, assim apenas 96 dos 128 *bits* foram utilizados para o acionamento das fontes de corrente e fontes de tensão.

Outro problema foi encontrado no conversor D/A de 8 bits, que funcionava apenas com 7 bits, fazendo com que a conversão não fosse linearmente monotônica.

O último problema verificado foi o não funcionamento de uma das fontes de corrente da gaussiana A, sendo necessário montar um circuito extra de forma discreta utilizando transistores bipolar, para não prejudicar o treinamento da rede.

Capítulo 5

RESULTADOS EXPERIMENTAIS OBTIDOS E APLICAÇÕES

Foram feitas duas aplicações através do treinamento da rede neural. Uma delas foi à aproximação de uma função senóide ideal e a outra uma linearização da saída de um VCO.

Através do treinamento da rede neural e comparando os resultados experimentais do projeto, notou-se através da figura 20 e figura 21 que o circuito com a RNBR tem capacidade de ser treinada.

5.1) Aproximação da função senóide

Para o cálculo do erro quadrático médio, fez-se a somatória da diferença ao quadrado entre a saída e a senóide ideal.

O erro quadrático médio (*fitness*) para a figura 20, foi de aproximadamente 0,05 com vetor de aquisição de 600 pontos.

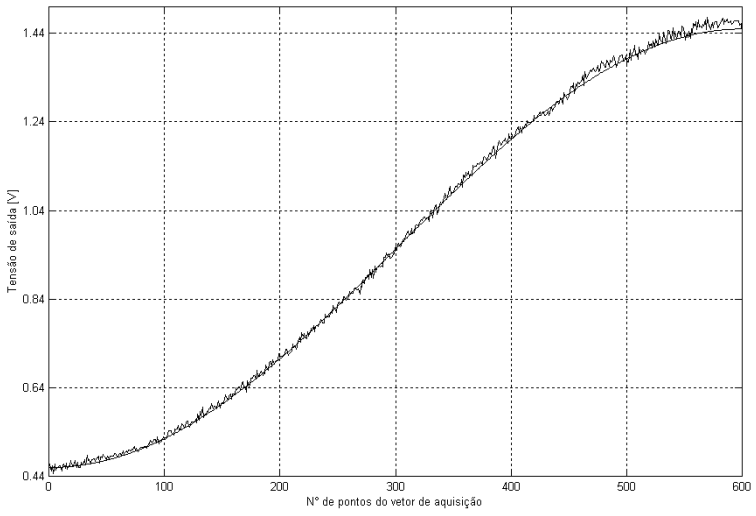

Figura 20 - Resultado do treinamento da RNFBR para uma senóide ideal de -90° a +90°.

O Algoritmo Genético Simples foi a base deste projeto por questões de conseguir boa convergência e robustez com um algoritmo simples.

De maneira que o AG pudesse surtir maiores eficiências de busca do mínimo local, os operadores genéticos (recombinação e mutação) foram aplicados não para o *string* de 128 bits como um todo, mas sim para cada 8 bits, responsável pelo acionamento de cada fonte de corrente e tensão individual. Com isso, comparando os resultados, temos que:

- Aplicando operadores genéticos para o *string* de 128 bits todo, nota-se que o número de interações necessárias para a convergência foi de aproximadamente 200 interações, equivalente a 50 minutos.
- Aplicando operadores genéticos a cada 8 *bits* do string de 128 bits, nota-se que o treinamento da RNFBR tornou-se mais robusto e rápida, fazendo com que o número de interações necessárias para a convergência fosse menor que 15 interações, equivalente a 5 minutos.

Aplicando o AG para os 128 *bits* e para o outro aplicando apenas à cada 8 bits dos 128 bits do string, nota-se que o tempo de convergência diminuiu bruscamente, na ordem de aproximadamente 92,5%, mantendo sua robustez e eficiência.

A configuração do treinamento da rede neural foi a seguinte:

Tamanho do vetor de aquisição = 600 pontos;
Número de interações = 300;
N° de indivíduos da população = 178;
Taxa de recombinação = 90%;
Taxa de mutação = 20%.

Com o treinamento da rede adquiriu-se o melhor indivíduo (peso) capaz de obter o melhor sinal de saída para a função senóide e este é apresentado na tabela 1.

Tabela 1 - *String* equivalente à melhor saída do treinamento da rede para a senóide ideal

Gaussianas	**A**	**B**	**C**	**D**
Amplitude	10101000	(Fonte Queimada)	00001111	00001010
Amplitude	10001110		11111010	00011111
Centro	00000111		01100001	00011011
Largura	00010011		01001110	10110100
	32 *Bits*		32 *Bits*	32 *Bits*

= 96 *Bits*

5.1.1) Aplicações: Gerador de Onda Triangular-Senoidal

A primeira aplicação da RNBR foi a aproximação de uma função senoidal. Na literatura, há alguns circuitos que podem ser utilizados para conversão de onda triangular-senoidal [3][4][5]. A RNBR proposta nesta tese pode ser considerada uma generalização desses circuitos, pois ela permite a aproximação ou síntese de praticamente qualquer função, desde que se disponha de um número suficientemente grande de funções de base. Desta forma, o circuito pode ser muito útil na implementação integrada ou discreta de qualquer função não-linear [2].

A função senoidal que se deseja aproximar é gerada pelo programa em linguagem C, onde será comparada com o sinal de saída do CI. O índice de comparação utilizado é o erro quadrático médio, onde a soma das diferenças pontuais ao quadrado é feita para todo o vetor de aquisição.

Após o treinamento da rede, obteve-se o peso da rede (*string* de 128 *bits*), onde este é armazenado em um arquivo de programa para ser inserido ao CI em futuras análises ou aplicações em conversores triangular-senoidal, ou seja, é o indivíduo que inserido no CI, é capaz de gerar o melhor resultado de saída, mesmo em *off-line*.

Na figura 21, é apresentado o funcionamento do conversor de onda triangular-senoidal para freqüência de 50 Hz.

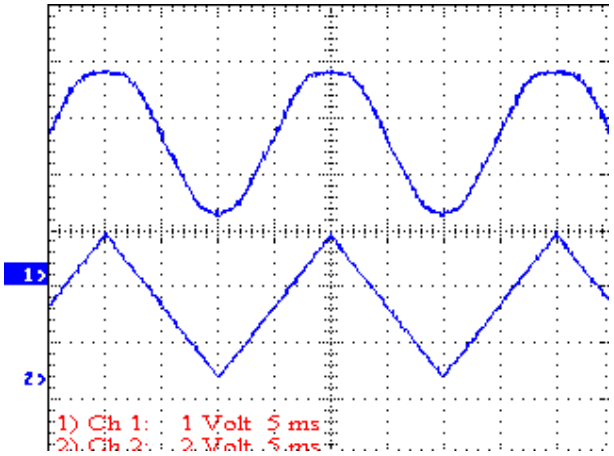

Figura 21 - Conversor triangular-senoidal em 50 Hz.

Testes foram realizados em altas freqüências (1 KHz e 10 KHz) apresentadas nas figuras 22 e 23 respectivamente e notou-se que o C.I funciona corretamente.

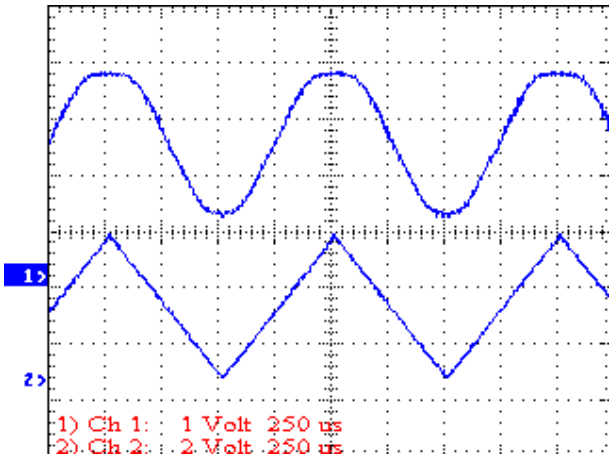

Figura 22 - Conversor triangular-senoidal em 1 KHz.

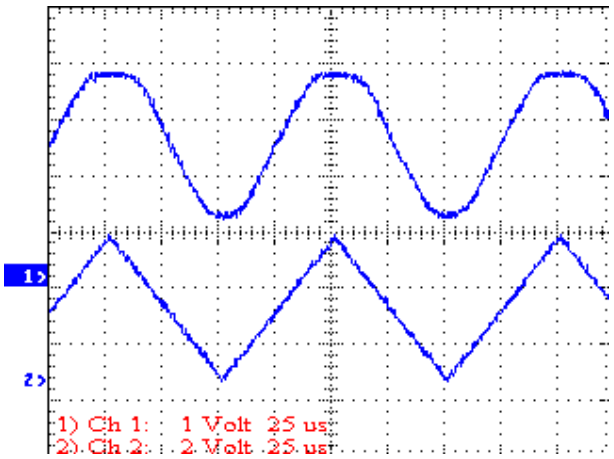

Figura 23 - Conversor triangular-senoidal em 10 KHz.

Através do osciloscópio digital onde foram feitas as medidas, analisou-se também na freqüência com auxílio da transformada rápida de Fourier (FFT) o sinal senoidal da figura 23. A figura 24 apresenta a FFT do sinal de saída com 50 Hz.

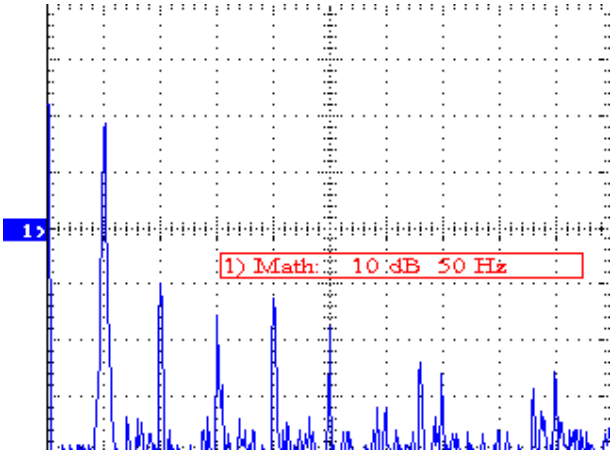

Figura 24 - FFT do sinal da figura 22.

A figura 24 mostra que o sinal possui várias distorções harmônicas, fazendo com que o sinal senoidal de saída do CI não seja perfeita. As possíveis causas destas distorções podem estar no número insuficiente de gaussianas para efetuar esta aproximação e também a baixa resolução de sua programação. As harmônicas em altas freqüências devem-se provavelmente ao fato de que o circuito de teste não tenha sido convenientemente blindado contra interferências eletromagnéticas.

Através do osciloscópio digital, foram feitas aquisições de dados das formas de ondas das 3 gaussianas geradas individualmente. Estes estão apresentadas nas figuras 25, 26 e 27.

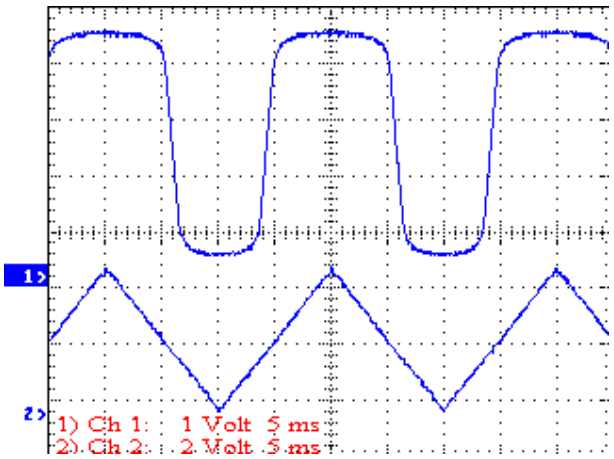

Figura 25 - Gaussiana A.

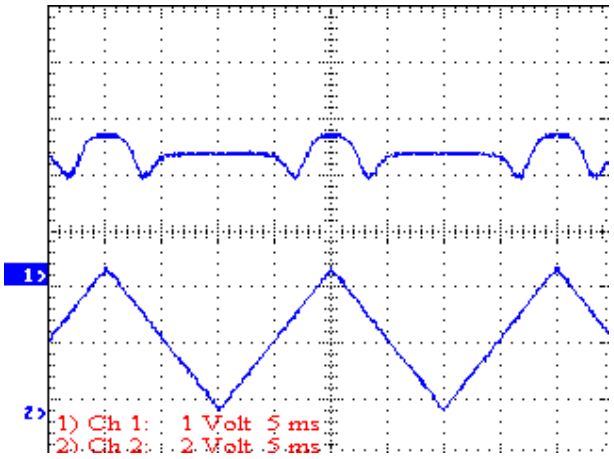

Figura 26 - Gaussiana C.

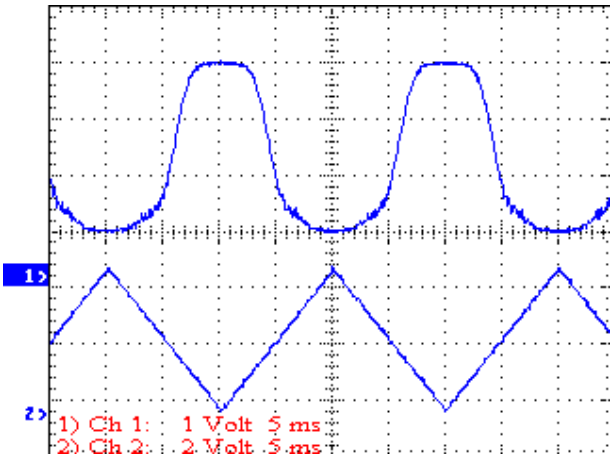

Figura 27 - Gaussiana D.

As figuras 25, 26 e 27, são as gaussianas individuais geradas através do CI, onde fazendo a somatória linear, gera o sinal de saída do conversor triangular-senoidal apresentado na figura 21, que é a finalidade em termos de aplicação deste projeto.

5.2) Linearização do VCO

A figura 28 apresenta o resultado do treinamento da rede neural para a linearização da saída do VCO. A reta inclinada na figura 28 indica a tensão de entrada (rampa) e os pontos indicam suas respectivas freqüências da tensão de entrada.

Para esta aplicação, nota-se que a rede neural com três gaussianas não foi robusta o suficiente para este tipo de treinamento, como pode ser observado a abrupcidade no sinal de saída do VCO. Vários testes foram feitos para verificar a não capacidade da rede em fazer o treinamento correto, alterando a taxa de recombinação e a taxa de mutação da rede neural, mas em todas as análises foram constatados este tipo de erro. Novamente a causa do erro deve-se provavelmente ao número limitado de funções gaussianas disponíveis e a baixa resolução da programação.

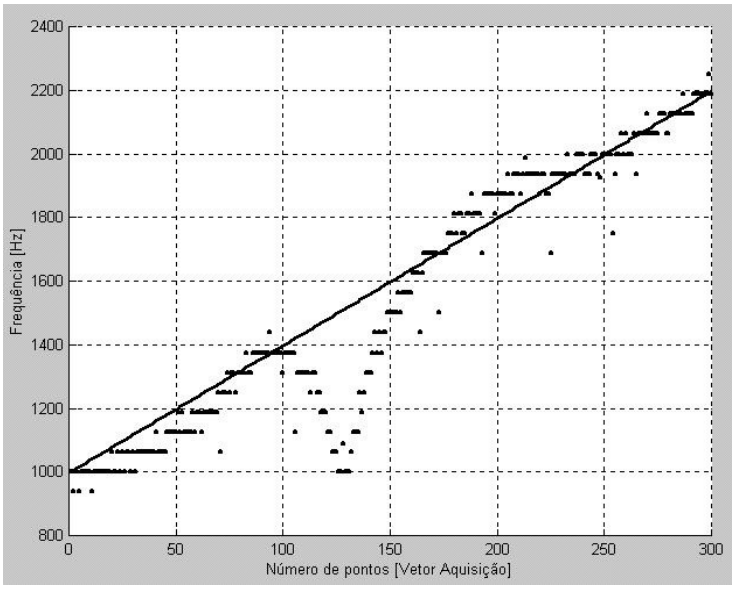

Figura 28 - Resultado do treinamento da RNFBR para a saída de um VCO.

Para este treinamento, utilizou-se um vetor de 300 pontos por ser um treinamento extenso como segue o fluxograma da figura 17.

Capítulo 6
CONCLUSÕES

Com os resultados obtidos experimentalmente, conclui-se que é possível realizar o treinamento de uma rede neural de base radial através de um *hardware*. Neste trabalho consistiu a implementação um *software* utilizando a computação evolucionária para realizar o treinamento de um circuito integrado com tecnologia BiCMOS 0.8μm da AMS que gerava funções de base radiais "gaussianas".

Duas aplicações foram apresentadas e em ambas notou-se que o treinamento da rede neural foi robusto e eficiente, pois conseguiu-se bons resultados. No treinamento para a linearização da saída do VCO, verificou-se um rendimento menor comparado ao treinamento da função senóide ideal pelo fato do CI gerar apenas três gaussianas. Uma solução para a minimização desse erro, seria aumentar o número de gaussianas, ou seja, aumentar o número de funções de base para que a rede neural tenha capacidade de aproximar qualquer função arbitrária, com um grau desejável de precisão.

Este trabalho mesmo sendo de simples elaboração, possuiu grandes dificuldades de ser implementado, pois necessitava de interfaceamento entre *software* e *hardware* para a realização do treinamento da rede neural de forma *online*.

O algoritmo genético utilizado no treinamento da rede neural, comportou-se de forma eficaz e robusta, por fazer parte da classe correspondente aos métodos probabilísticos de busca e otimização, apesar de não ser aleatório e também porque AG tenta direcionar a busca para regiões onde é provável que os pontos ótimos estejam.

Como sugestão de continuidade deste trabalho pretende-se implementar uma memória não-volátil e reprogramável para que os pesos obtidos através do treinamento não sejam perdidos e também aumentar a resolução do conversor digital analógico para 10 bits, o que provavelmente melhorará o desempenho dos circuitos propostos nesta dissertação.

Referências Bibliográficas

[1] M. B. Lucks and N. Oki, “An analog implementation of radial basis function network appropriate for transducer linearizer”, 42nd Midwest Symposium on Circuits and Systems, Volume: 2,2000, Page(s): 1109-1112 vol.2, Aug.2000.

[2] J. P. Oliveira, “Implementação de uma rede neural de base radial utilizando tecnologia BiCMOS”, Dissertação de mestrado na faculdade de engenharia de Ilha Solteira, Pagina 9, Maio de 2002.

[3] S. Franco. Design with Operational Amplifiers and Analog Integrated Circuits. 2nd Edition, Boston, McGraw-Hill, 1998.

[4] TURCHETTI, C and CONTI M., “A General Approach to Nonlinear Synthesis with MOS Analog Circuits”, IEEE Trans. On Circuits and Systems-I: Fundamental Theory and Applications, vol.40, n°.9, September, 1993.

[5] TOUMAZOU C., LIDGEY F. J. and HAIGH D. G., “Analog IC Design: The Current-Mode Approach”, IEE Circuits and Systems Series, 1990.

[6] Y. Davidor. "Genetic algorithms: a survey". Dynamic, Genetic, and Chaotic Programming, B. Soucek and The IRIS Group (eds.), John Wiley & Sons, Inc., New York, USA, 1992, pp.323-338.

[7] J.M. Barreto. "Conexionismo e a Resolução de Problemas", Titular Professor Contest Dissertation, Departamento de Informatica e Estatística, UFSC, Florianópolis, SC, 1996.

[8] D.J. Jason and J.F. Frenzel. “Training product unit neural networks with genetic algorithms. IEEE Expert, vol.1, no.1, pp.26-33, 1993.

[9] H. Muhlenbein. “The Dynamics of Evolution and Learning - Towards Genetic Neural Networks”. German National Reserach Center for Computer Science, 1989.

[10] D.L. Prado. “New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques”. Electronics Letters, vol.28, no.16, pp.1560-1561, 1992.

[11] V.W. Porto, D. Fogel, and L. Fogel. “Alternative neural networks training methods”. IEEE Expert, vol.10, no.3, pp.16-22, June 1995.

[12] S. Austin. "An introduction to genetic algorithms". AI Expert, vol.1, no.1, pp. 48-53, 1990.

[13] J. Dias. “Treinamento Híbrido de Redes Neurais para Processamento de Informações Biomédicas”. Tese de Doutorado, Universidade Federal de Santa Catarina, UFSC, Brasil, 1999.

[14] T. Kozek and T. Roska and L.O. Chua. "Genetic algorithm for CNN template learning", IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications, vol.40, no.6, pp.392-402, 1993.

[15] D. Park and A. Kandel. "Genetic-based new fuzzy reasoning models with application to fuzzy control". IEEE Transactions on Systems, Man, and Cybernetics, vol.24, no.1, pp.39-47, 1994.

[16] M. Srinivas and L.M. Patnaik. "Adaptive probabilities of crossover and mutation in genetic algorithm". IEEE Transactions on Systems, Man, and Cybernetics, vol.24, no.4, pp.656-667, 1994.

[17] H. Garis. "Artificial embryology: the genetic programming of an artificial embryo", in Dynamic, Genetic, and Chaotic Programming, B. Soucek and The IRIS Group (eds.), John Wiley & Sons, Inc., New York, USA, 1992, pp.373-393.

[18] A.S. Algarve. “Simulação do Sistema Circulatório com Controle Neuronal Central", Relatório Interno - GPEB, UFSC, Florianópolis, Brasil, 1997.

[19] B. Soucek and The IRIS Group. Neural and Intelligent Systems Integration: Fifth and Sixth Generation Integrated Reasoning Information Systems. John Wiley & Sons, Inc., New York, USA, 1991.

[20] LUCKS M. B, “Implementação Eletrônica de uma Rede Neural Artificial com Função de Base Radial e suas Aplicações”, Dissertação de mestrado na faculdade de engenharia de Ilha Solteira, Dezembro de 1998.

[21] MALVINO A. P., “Eletrônica – Volume 2”, McGraw-Hill, 1986.

[22] D.B. Fogel. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. New York,USA: IEEE Press, 1995.

[23] T. Bäck, U. Hammel, and H. Schwefel. “Evolutionary computation: comments on the history and current state”. IEEE Transactions on Evolutinary Computation, vol.1, no.1, pp. 3-17, April 1997.

[24] J.H. Holland. Adaptation in Natural and Artificial Systems, Cambridge, Massachusetts: MIT Press, 1975.

[25] C. Darwin. The Origin of Species. Fac-Simile da edição original – Charles W. Eliot, L. L.D. – 1981.

[26] L.C. Dunn and T.H. Dobzhansky. Hereditary, race and Society. Mentor Books – The New American Libray, 4a. ed., 1950.

[27] D.E. Goldberg. "Genetic Algorithms in Search, Optmization, and Machine Learning". Addison-Wesley Publishing Company, Inc., Reading, Massachussets, 1989.

[28] M.C. South and G.B. Wetherill and M.T. Tham. “Hitch-hiker's guide to genetic algorithms". Journal of Applied Statistics, vol.20, no.4, pp.153-175, 1993.

[29] J. Tanomaru. “Motivação, fundamentos e aplicações de algoritmos genéticos". In Proc. do II Congresso Brasileiro de Redes Neurais, vol.1, no.3, Curitiba, Brasil, 1995, pp.331-411.

[30] N. Paulino and J. E. Franca, “A CMOS digitally programmable current multiplier” ISCAS – Internacional Symposium of Circuits and Systems, Vol.1, pp 254-257, 1996.

APÊNDICE A

PORTA PARALELA

A1) Introdução

A porta paralela é uma interface de comunicação entre o computador e um periférico. Quando a IBM criou seu primeiro PC (Personal Computer) ou Computador Pessoal, a idéia era conectar a essa Porta uma impressora, mas atualmente, são vários os periféricos que utilizam-se desta porta para enviar e receber dados para o computador (exemplos: Scanners, Câmeras de vídeo, Unidade de disco removível e outros).

Linguagem de programação como: C/C++/C++Builder, Pascal/Delph ou mesmo o Visual Basic, desenvolve programa que pode controlar um aparelho conectado à porta paralela, ou um programa de transferência de arquivos entre dois computadores, utilizando um cabo paralelo como meio de transmissão.

A2) Modelos de Porta Paralela

- Transmissão Unidirecional: A porta paralela SPP (Standard Parallel Port) pode chegar a uma taxa de transmissão de dados a 150KB/s. Comunica-se com a CPU utilizando um BUS de dados de 8 bits. Para a transmissão de dados entre periféricos são usado 4 bits por vez.

- Transmissão Bidirecional: A porta avançada EPP (Enhanced Parallel Port) chega a atingir uma taxa de transferência de 2 MB/s. Para atingir essa velocidade, será necessário um cabo especial. Comunica-se com a CPU utilizando um BUS de dados de 32 bits. Para a transmissão de dados entre periféricos são usado 8 bits por vez. A porta avançada ECP (Enhanced Capabilities Port) tem as mesmas características que a EPP, porém, utiliza DMA (acesso direto à memória), sem a necessidade do uso do processador, para a transferência de dados. Utiliza também um buffer FIFO de 16 bytes.

A3) Endereço da Porta Paralela

O computador nomeia as Portas Paralelas, chamando-as de LPT1, LPT2, LPT3 etc, mas, a Porta física padrão de seu computador é a LPT1, e seus endereços são: 378h (para enviar um byte de dados pela Porta), 378+1h (para receber um valor através da Porta) e, 378+2h (para enviar dados). Às vezes pode estar disponível a LPT2, e seus endereços são: 278h, 278+1h e 278+2h, com as mesmas funções dos endereços da porta LPT1 respectivamente, como apresentado na tabela 2.

Tabela 2: Endereçamento da porta LPT1 e LPT2.

Nome da porta	Endereço da memória	Endereço da porta	Descrição	
LPT1	0000:0408	378 hexadecimal	888 decimal	Endereço base
LPT2	0000:040A	278 hexadecimal	632 decimal	Endereço base

A4) Registradores

Utilizando a porta paralela conectada a uma impressora, os endereços terão nomes sugestivos, como segue abaixo na tabela 3.

Tabela 3: Endereço de registro e descrição da LPT1 e LPT2.

Nome	Endereços LPT1	Endereços LPT2	Descrição
Registro de dados	378h	278h	Envia um byte para a impressora
Registro de status	379h	279h	Ler o Status da impressora
Registro de controle	37Ah	27Ah	Envia dados de controle para a impressora

A5) O conector DB25

O DB25 é um conector que fica na parte de trás do gabinete do computador, e é através deste, que o cabo paralelo se conecta ao computador para poder enviar e receber dados.

No DB25, um pino está em nível lógico 0 quando a tensão elétrica no mesmo está entre 0 à 0,4v. Um pino se encontra em nível lógico 1 quando a tensão elétrica no mesmo está acima de 3.1 e até 5v.

A figura 29 abaixo mostra o conector padrão DB25, com 25 pinos, onde cada pino tem um nome que o identifica e este é mostrado nas figuras 30 e 31.

Figura 29 - Foto do conector DB25 macho

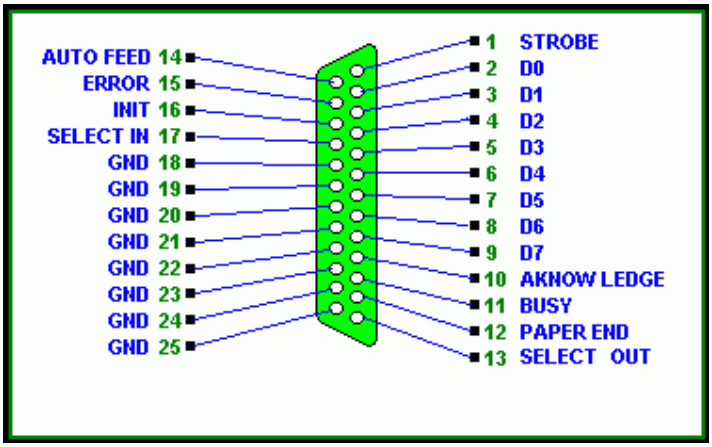

Figura 30 - Significado de cada pino do DB25

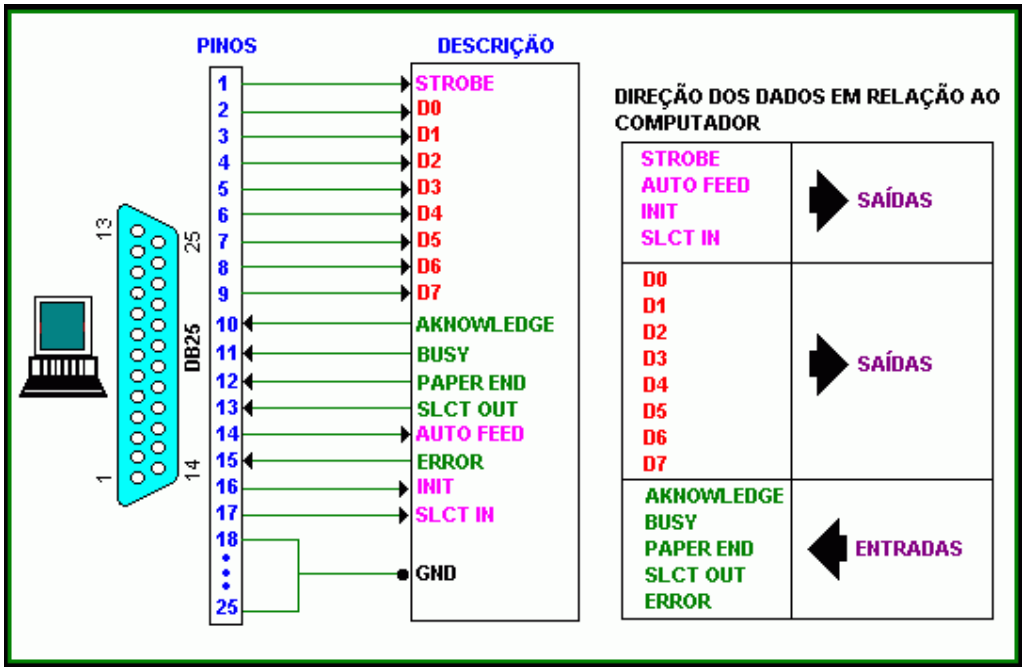

Figura 31 - Esquema de funcionamento do DB25 no modo SPP

A6) O conector macho centronics 36 pinos

O conector macho centronics 36 pinos faz parte do cabo da impressora, é através deste cabo que a impressora é conectada ao computador.

Quando desenvolvemos um projeto que utilize uma interface para conectarmos ao computador, poderemos utilizar um conector centronics 36 pinos fêmea, isso faz com que nossa interface aproveite o cabo da impressora, onde poderemos conseguir com facilidade em lojas de Informática.

A figura 32 mostra o conector centronics 36 pinos.

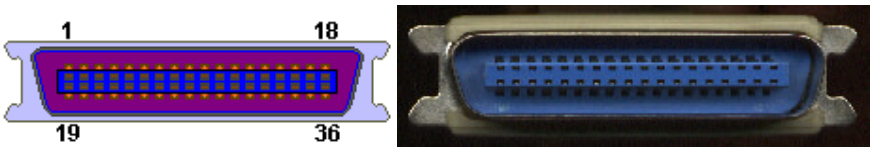

Figura 32 - Conector centronics 36 pinos

Somente os pinos de 2 a 9 são utilizados neste projeto, que são os pinos onde serão enviados os string's de 128 bits do treinamento da rede neural e um dos pinos 19 à 30 que são os GND's.

A7) Funcionamento e teste iniciais

Primeiramente foi implementado um circuito para testar o funcionamento correto da porta paralela e seu controle, relacionando aos parâmetros de funcionamento do CI. Assim, montou-se um circuito da seguinte maneira, como apresentado na figura 33.

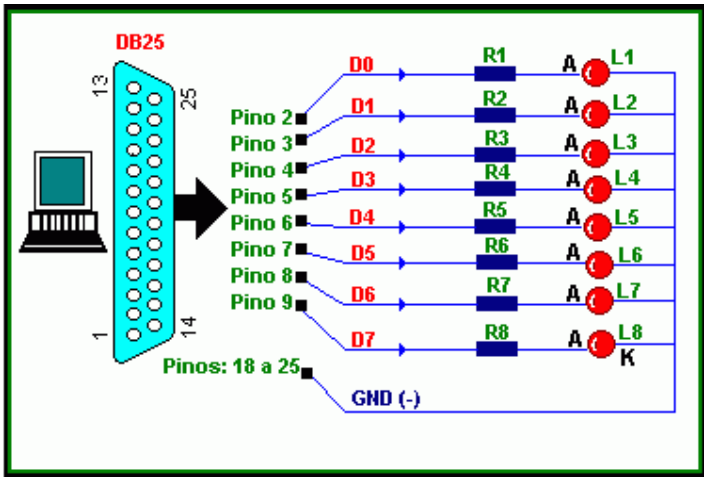

Figura 33 - Esquema do circuito de teste da porta paralela, com led's

A princípio o objetivo do circuito da figura 34, é ligar e desligar oito LED's, conectados através de um cabo à porta paralela. O terminal do catodo (K) dos LEDs estão ligados aos terminais dos resistores, que estes por si, estão ligados através do cabo aos pinos do DB25.

O programa feito em linguagem C pode ser compilado e executado para acender e apagar os LEDs do circuito da figura 33.

/ Envia sinal para a Porta Paralela LPT1: */*

```
#include <stdio.h>
#include <dos.h>
#define LPT1 0x378
int main(void)
{
   unsigned char Valor=128; //Em binário: 10000000
   while( Valor > 0 )
  {
          outportb(LPT1, Valor); // Envia para a Porta LPT1
          printf("\nPressione uma tecla para ascender o próximo LED...");
          getch( );
          Valor = Valor >> 1; //A cada passagem, o bit 1 é movido para a direita
   }
}
```

Este programa envia à porta paralela oito bits, um a cada vez. A sequência de bytes geradas é vista na tabela 4:

Tabela 4: Seqüência de bits enviados para a porta paralela.

Decimal	Hexadecimal	Binário	Pino/Fio ativo (5V)	Comentário
128	80	10000000	9-D7	Cada bit do byte enviado à Porta Paralela está relacionado com um pino do DB5, e um fio do cabo paralelo, fisicamente. Ao enviar um byte, que o(s) bit(s) esteja(m) ligado(s) ou desligado(s), os LEDs acende(rão) ou apaga(rão) conforme os estados dos bits.
64	40	01000000	8-D6	
32	20	00100000	7-D5	
16	10	00010000	6-D4	
8	8	00001000	5-D3	
4	4	00000100	4-D2	
2	2	00000010	3-D1	
1	1	00000001	2-D0	

Para saber como o computador agrupa os bits num byte, observe o esquema da tabela 5.

Tabela 5: Agrupamento dos bits em um byte.

No esquema acima observe que cada nibble equivale a 4 bits e a contagem dos bits é feita da direita para a esquerda (0,1,2,3...).

APÊNDICE B

PLACA DE AQUISIÇÃO DE DADOS

B1) Introdução

A placa de aquisição de dados tem como finalidade captar sinais oriundos de sistemas ou hardwares com o objetivo de analisar e/ou trabalhar com tais sinais, fazendo assim análises mais apuradas dos mesmos.

Os conversores de 12 bits fazem a correspondência de 1096 valores binários com a faixa de tensão, de 0 a 5V, representando uma faixa de tensão de 1,22mV. O conversor de 12 bits representa a grandeza analógica na entrada ou saída melhor que uma conversor de 8 bits.

Neste apêndice será abordado a placa de aquisição de dados de interface comercial com conversores A/D e D/A de 12 bits.

Esta placa é ligada e instalada diretamente no PC com software provido do kit. O modelo utilizado para este projeto é a PCL-711B (PC – Multlab Card) da Lab. & Engineering Add-on's for PC/XT/AT.

B2) A placa PCL-711 PC-MULTILAB

A placa PCL-711 é uma interface genérica para aquisição de dados e controle, compatível com microcomputadores IBM PC/XP/AT.

A placa tem um conjunto de chaves (dip switch) que formam um endereço-base. As chaves são enumeradas de 1 a 6, cada qual ligada a um bit do barramento de endereços, de A9 a A4 respectivamente.

O fato de não se utilizar os bits de A3 a A0 deve-se a características da placa que utiliza uma faixa de 16 endereços, contados a partir do endereço-base selecionado pelas chaves. O endereço-base escolhido deve estar livre de outros dispositivos. Os endereços-base normalmente utilizados são 220H ou 300H.

B2.1) Configurações

- 8 entradas analógicas com 12 bits de resolução para o conversor A/D;
- A conversão pode ser inicializada por software, temporizador programável ou por um sinal externo;
- Nível de interrupção IRQ programável para transferência de dados A/D;
- 1 canal de saída D/A com resolução de 12 bits com faixa de saída de 0-5V ou 0-10V;
- 16 entradas digitais;
- 16 saídas digitais;
- Programável em Basic, Pascal, C e C++.

B2.2) Especificações da entrada analógica (A/D)

Canais: 8 entradas

Resolução: 12 bits, conversão mediante aproximação sucessivas;

Faixa de entrada: programável por software a: ±5V, ±2,5V, ±1,25V, ±0,625V e ±0,3125V;

Conversor: AD574 ou equivalente;

Tempo de conversão: 25ms Max;

Não Linearidade: ±1 bit;

Modo de disparo: por software, por temporizador programável ou por um sinal externo;

Transferência de dados: por software ou por interrupção;

Nível de IRQ: IRQ2 a IRQ7;

Tensão máxima: ±30V contínuo.

B2.3) Especificações da saída analógica (D/A)

Canais: 1 saída analógica;

Resolução: 12 bits;

Faixa de saída: 0-5V ou 0-10V;

Conversor: PM7548GP ou equivalente;

Tempo de estabilização: 30ms;

Capacidade de saída: ±5mA Max;

Tensão de referência: Interna –5V e –10V (±0,05V);

Não Linearidade: ±1/2 LSB

B2.4) Entrada Digital

Canais: 16 bits, compatível ao TTL;

Tensão de entrada: Baixo – 0,8V max e alto – 2,0V min;

Corrente de entrada: Baixo – 0,4mA max @0,5V max e alto – 0,05mA max @2,7Vmin;

B2.5) Saída Digital

Canais: 16 bits, compatível ao TTL;

Tensão de saída: Baixo – 8mA @0,5V max e alto – 0,4mA @ 2,4V min.

B2.6) Seleção de endereços de I/O

A placa de aquisição é controlada através das portas de I/O via PC. Estes endereços de I/O são apropriados para não ocorrer conflitos entre os dispositivos instalados no PC, com relação ao PLC-711B.

Os endereços bases da porta de I/O são selecionados através de 6 posições da chave SW1, na própria placa.

Os endereços válidos são de 000 à 3F0 (hexadecimal) e estão apresentados na tabela 6. O endereço padrão é 220.

Tabela 6 – Endereços bases da porta de I/O

Endereço de I/O (Hexadecimal)	Posição da chave (SW1)					
	1	2	3	4	5	6

	A9	A8	A7	A6	A5	A4
000-00F	0	0	0	0	0	0
100-10F	0	1	0	0	0	0
...						
200-20F	1	0	0	0	0	0
210-22F	1	0	0	0	0	1
220-22F *	1	0	0	0	1	0
…						
300-30F	1	1	0	0	0	0
3F0-3FF	1	1	1	1	1	1

Observação: 0 = Ligado e 1 = Desligado

* = Endereço padrão

A saída D/A da placa de aquisição depende da tensão de referência seleciona através do *"jumper"*, JP1. A tensão de referência pode ser de 0 a +5V ou de 0 a +10V.

Para aplicação deste projeto utilizou-se a tensão de referência de 0 a +5V.

B3) Layout da placa de aquisição

O layout da placa PCL-711 é apresentado na figura 34.

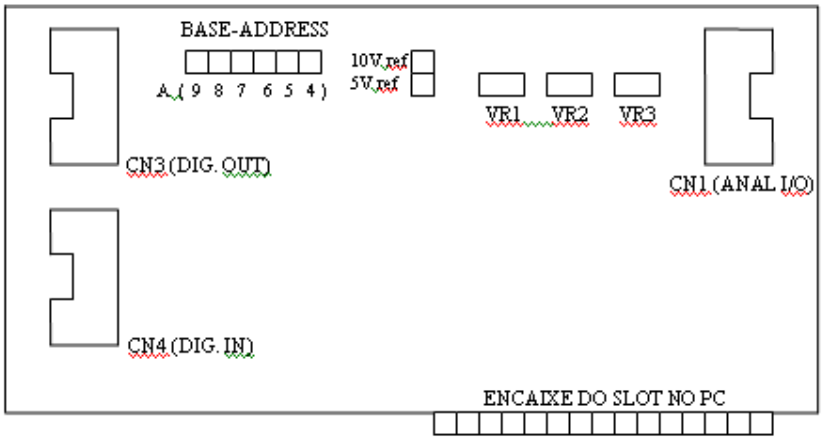

Figura 34 Layout da placa PLC -711.

A placa é colocada diretamente sobre o *slot* e através de um flat cable conctados em CN1, CN3 e CN4 têm-se o acesso aos conversores e aos portos;

CN1 - é um conector para entrada de sinais analógicas (conversor A/D) e saída de sinal analógico (conversor D/A);

CN3 - é um conector para saída de sinais digitais;

CN4 - é um conector para entrada de sinais digitais;

Observação: Os conectores CN3 e CN4 não são utilizados neste trabalho. Somente CN1, com a utilização dos conversores A/D e D/A.

JP - seleciona uma das duas faixas de tensão analógica do conversor D/A. Se o jumper estiver em 5V ref a faixa de tensão de saída será de 0 a 5 V e se o jumper estiver em 10V ref a faixa será de 0 a 10V;

VR1 – potenciômetro de ajuste do ganho no conversor D/A;

VR2 - potenciômetro de ajuste do offset do conversor A/D;

VR3 - potenciômetro de ajuste do ganho no conversor A/D.

Com relação ao conector CN1, os pinos A/D0 a A/D7 são entradas analógicas do conversor A/D e D/A é a saída analógica do conversor D/A. A.GND são os terras analógicos da placa.

B4) O Conversor A/D

O conversor A/D da placa PCL-711é o CI AD574, de 12 bits. Há a multiplexação de sinais analógicos na entrada, ou seja, pode-se conectar oito sinais analógicos distintos, entretanto, o conversor realiza apenas uma conversão por vez. A tensão do sinal de entrada deve estar entre -5V e +5V. Para a tensão de -5V o número digital resultante da conversão é 0000H e para 5V o resultante será o hexadecimal 0FFFH.

Esta escala está representada na figura 35, onde V é uma tensão analógica qualquer e Npc sua representação em numérica.

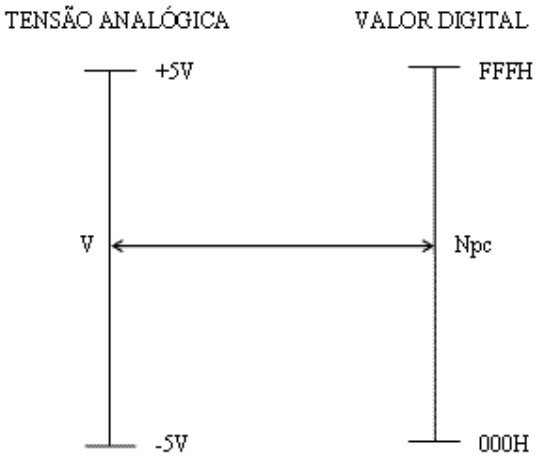

Figura 35 – Relação Tensão – Número Hexadecimal.

Utilizando uma regra de três simples:

$$\frac{V-(-5)}{5-(-5)} = \frac{Npc-0}{4095-0}$$

As tensões esta em base decimal.

Isolando V da equação anterior resultará:

$$V = \frac{10 \times Npc}{4095} - 5$$

A máxima sobretensão suportável pelo AD574 é de +30V.

O conversor A/D usa os endereços base+4 para os bytes menos significativos e base+5 para o byte mais significativo, com a seguinte configuração de bits da tabela 7.

Tabela 7: Configuração de bits dos registradores do conversor A/D.

Endereço	BITS DO REGISTRADOS							
BASE+4	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
BASE+5	X	X	X	DY	AD11	AD10	AD9	AD8

X – representa qualquer valor (0 ou 1);

AD0 a AD11 – é o equivalente binário da tensão analógica;

DY – é o sinal de fim de conversão; se a conversão não acabou, DY = 1, caso contrário, DY = 0.

A entrada analógica selecionada deve ser ativada por uma operação de escrita no endereço base +10 (MUX SCAN CHANNEL), com o byte de dados a ser enviado como apresentado na tabela 8.

Tabela 8 – Configuração dos bits do registrador do multiplexador.

Endereço	BITS DO REGISTRADOS						
BASE+10	X	X	X	X	CL2	CL1	CL0

Os sinais CL2, CL1 e CL0 selecionam a entrada analógica de acordo com a tabela 9.

Tabela 9 – Bits de seleção e entrada analógica selecionada.

Seleção			Entrada Ativa
CL2	CL1	CL0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Para realizar uma conversão A/D deve-se seguir os seguintes passos:

A – Conectar o sinal na entrada analógica escolhida;

B – Selecionar a entrada escolhida: envie ao endereço base+10 o byte correspondente a entrada escolhida. Este byte é determinado pela tabela X3 e tabela X4;

C – Aplicar o sinal de partida: envie ao endereço base +12 (sinal de partida) qualquer valor;

D – Ler o conteúdo do byte mais significativo: leia o conteúdo do endereço base +5, isolando o bit D4 do mesmo; se este for 1, deve-se fazer nova leitura; caso contrário, faça o passo seguinte;

E – Ler os bytes superiores e inferiores: leia o conteúdo do endereço base +5 e após, o conteúdo do endereço base +4;

F – Converter os valores binários lidos em valores inteiros.

B5) O Conversor D/A

A placa de aquisição possui um canal de conversor D/A de 12 bits. Os registradores do conversor D/A usam os endereços base+4 e base+5. DA0 é o bit menos significativo e DA11 o mais significativo. O formato do dado em cada endereço é apresentado na tabela 10.

Tabela 10 – Configuração de bits dos registradores do conversor D/A.

Endereço	BITS DO REGISTRADOR							
BASE+4	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
BASE+5	X	X	X	X	DA11	DA10	DA9	DA8

Os endereços dos conversores A/D e D/A são idênticos; o que os diferem é o tipo de operação a realizar: escrita para conversores D/A e leitura para conversores A/D.

Quando se utilizada o conversor D/A o byte menos significativo (endereço base+4) deve ser escrito primeiro e depois o byte mais significativo (endereço base+5). O jumper JP colocado na posição 5V ref faz com que a tensão na saída possa variar de 0 a 5V e na posição 10V ref, de 0 a 10 V.

APÊNDICE C

Softwares desenvolvido para o treinamento da rede neural de base radial

I) Função Senoidal – Conversor Triangular-Senoidal

```
/* Circuito integrado de uma Rede de Base Neural Radial */
/* Utiliza 3 Gaussianas */
/* Bit 1 - Dados, bit 2 - Clock, Bit 3 - Reset e bit 4 - Clock master */
/* o Chip , resetado quando o reset , igual ao nivel baixo */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <dos.h>

#define DELAY 1
#define NM 3
#define LINHA 73 /* 75 numero da populacao */
#define NP 600 /* 1000 tamanho do vetor */
#define INTER 800 /* 200 numero de interacoes */
#define RECOMB 90 /* taxa de recombinacao 90% */
#define MUTA 5 /* taxa de mutacao 2% */

/* variaveis do programa principal */

int parada;

/* variaveis do vetor handomico */

int cont1,y;        /* y(colunas) */
int h[128];   /* vetor tamanho 128 */

/* variaveis dente de serra */

int a,b,m,i,j;
float v[NP]; /* Vetor tensao */

/* variaveis do vetor senoide ideal */

int cont2;
float xe,kk,senoide;
float s[NP];

/* variaveis de acionamento */
```

```
int reset;
int clk_m=0;
int clk;
int p,jj,tt,f,c,saida;
int caracter[16];

/* variaveis compara */

int cont3,cont4,dd,yy;
float soma;
float dif[NP]; /* vetor diferenca entre a senoide e o norm */
float rk[LINHA]; /* vetor q guarda o erro quadratico de cada string */
int ini[LINHA][128]; /* matriz populacao inicial */

/* variaveis incumbente */

int cont13;
int best[128]; /* melhor string */
float inc,incumb;

/* variaveis genetico */

int rec1[128],rec2[128],rec3[128],rec4[128];
int p1,x,x1,x2,x3,rod,cont6,cont7,cont8,cont9;
float erro,erro1,inc,incumb;

/* variaveis envio2 */

int cont10,cont11,cont12,cont14;

/*******************************/
/*   PROGRAMA PRINCIPAL        */
/*******************************/

void main()
{
clrscr();

/* para resetar o CI logo no inicio do treinamento */
/* isso eh feito apenas 1 vez */

reset = 0;
saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[16]&0x01);  /* Deslocamento dos
bits */
outportb(0x378,saida);
printf("Sistema Resetado!!!\n\n");
printf("Tecle algo para iniciar o treinamento da Rede!!!");
getch();
```

```
inc=10000;
cont1=0;
cont3=0;
cont4=-1;
cont8=-2; /* linha par da matriz corrente */
cont9=-1; /* linha impar da matriz corrente */
cont11=0; /* linha da matriz corrente */
cont14=0; /* contador de parada das interacoes */

clrscr();

string(); /* chama a subrotina geracao de string 128 bits aleatorios */
envio(); /* envia os strings para o CI, gera a rampa e faz a aquisicao do sinal de saida do
CI */
compara(); /* compara os sinais e define o fitnes da populacao inicial */

} /* Fim do Programa Principal */


/***********************************/
/* Subrotina string 128 bits       */
/***********************************/

string()
{

/* srand(cont1); */  /* altera o peso dos numeros randomicos */
/* cont1=cont1+1; */  /* o normal eh +1 */

for (y=0;y<128;y++) /* varre 128 colunas */
        {
                h[y]=random(2);  /* cria a vetor randomico */
        }
for (y=8;y<16;y++) /* iguala os 8 primeiros bits com os 8 segundos bits */
        {
                h[y]=h[y-8];
        }
for (y=72;y<80;y++)
        {
                h[y]=h[y-8];
        }
for (y=104;y<112;y++)
        {
                h[y]=h[y-8];
        }
printf("\nString de 128 bits gerado aleatoriamente:\n");
for (y=0;y<128;y++)
        {
                printf("%d",h[y]);  /* imprime os strings correto */
        }
```

```
printf("\n");

}


/*****************************/
/* Envio do string para o CI */
/*****************************/

envio()
{

reset=1;
clk_m=0;

/* Leitura dos strings da matriz de 16 em 16 bits */

tt=-16;
f=0;
while (f<128)
        {
                tt=tt+16;
                f=f+16;
                p=-1;

                for (c=tt;c<f;c++)
                        {
                                p=p+1;
                                caracter[p]=h[c];
                        }
                for (jj=0;jj<16;jj++)
                        {
                                /* clock em baixa */
                                clk = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                                delay(DELAY);  /* tempo de acionamento dos leds */

                                /* clock em alta */
                                clk = 1;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                                delay(DELAY);

                                /* clock em baixa */
                                clk = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
```

```
                                outportb(0x378,saida);
                                delay(DELAY);

                                /* clock master em alta */
                                clk_m = 1;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                                delay(DELAY);

                                /* clock master em baixa */
                                clk_m = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                                delay(DELAY);

                                /* enviar 02 */
                                caracter[jj]=caracter[jj]>>1;
                                outportb(0x378,saida);
                                delay(DELAY);
                        }
        }
serra();
seno();
}


/*****************************/
/* Subrotina Dente de Serra */
/*****************************/

serra()
        {
                m=-1;
                outportb(0x30A,0x00); /* selecao entrada analogica */
                for(j=0;j<=0x0f;j++) /* gera os bs */
                        {
                                for (i=0;i<=0xff;i=(4095./NP)+i)
                                {
                                m=m+1; /* Tamanho vetor rampa */
                                outportb(0x304,i);
                                outportb(0x305,j);
                                outportb(0x30c,0x00); /* Sinal de partida */
                                for(a=0x10;a==0x10;) /* Espera pelo fim de conversao */
                                        {
                                                a=inportb(0x305);
                                                a=a & 0x10;
                                        }
                                a=inportb(0x305);
```

```
                        b=inportb(0x304);
                        a=b+a*256;
                        v[m]=(4.*a/4095-2);
                        }
                }
        salva();
    }

salva()
    {
        FILE *outfile;
        if((outfile = fopen("tensao.c","wb")) == NULL)
            {
                perror("Erro em criar tensao.c");
                exit(1);
            }
        for (m=0;m<NP;m++)
            {
                fprintf(outfile,"%f\n",v[m]);
            }
        fclose(outfile);
    }


/**********************/
/* Funcao Seno Ideal */
/**********************/

seno()
    {
        cont2=-1;
        for (kk=-90;kk<90.001;kk=(180./NP)+kk) /* range pontos */
            {
                xe=(3.14159265359/180.)*kk;
                senoide=sin(xe)+1; /* com offset de 1 volt */
                senoide=(0.99*senoide+0.92)/2;
                cont2=cont2+1;
                if(cont2<NP)
                    {
                        s[cont2]=senoide;
                    }
            }
    salva1();
    compara();
    }

salva1()
    {
        FILE *outfile;
        if((outfile = fopen("sen.c","wb")) == NULL)
```

```
                {
                        perror("Erro em criar sen.c");
                        exit(1);
                }
        for (cont2=0;cont2<NP;cont2++)
                {
                        fprintf(outfile,"%f\n",s[cont2]);
                }
        fclose(outfile);
}

/*************************************************/
/* Compara o sinal de saida com uma senoide ideal */
/*************************************************/

compara()
        {
                soma=0;
                for (dd=0;dd<NP;dd++)
                        {
                                dif[dd]=s[dd]-v[dd];
                                dif[dd]=pow(dif[dd],2.0);
                                soma=soma+dif[dd];
                        }
        /*      printf("String armazenado na matriz pop!!\n\n"); */
        /*      getch();  */
                rk[cont3]=soma;
                cont3=cont3+1;
                cont4=cont4+1;
                for (yy=0;yy<128;yy++)
                        {
                                ini[cont4][yy]=h[yy];
                        }
        if (cont4==LINHA-1)
                {
                        cont1=0;
                        incumbente();
                        genetico();
                }
        else
                {
                        string();
                        envio();
                }
        } /* fim da subrotina compara */



/**************/
/* Incumbente */
```

```
/**************/

incumbente()
        {

                for (cont6=0;cont6<LINHA;cont6++)
                        {
                                incumb=rk[cont6];
                                if (incumb<inc)
                                        {
                                                inc=incumb;   /* incumbente */
                                                for (cont13=0;cont13<128;cont13++)
                                                        {

        best[cont13]=ini[cont6][cont13];
                                                        }
                                                salva2();
                                                salva4();
                                        /*      serra2(); */
                                                }
                        }
}

salva2()
        {
                FILE *outfile;
                if((outfile = fopen("best.c","wb")) == NULL)
                        {
                                perror("Erro em criar best.c");
                                exit(1);
                        }
                for (cont2=0;cont2<128;cont2++)
                        {
                                fprintf(outfile,"%d\n",best[cont2]);
                        }
                fclose(outfile);
        }

salva4()
{
        FILE *outfile;
        if((outfile = fopen("inc.c","wb")) == NULL)
                {
                        perror("Erro em criar inc.c");
                        exit(1);
                }
                fprintf(outfile,"%f",inc);
                fclose(outfile);
}
```

```
serra2()
	{
		m=-1;
		outportb(0x30A,0x00); /* selecao entrada analogica */
		for(j=0;j<=0x0f;j++) /* gera os bs */
			{
				for (i=0;i<=0xff;i=(4095./NP)+i)
				{
				m=m+1; /* Tamanho vetor rampa */
				outportb(0x304,i);
				outportb(0x305,j);
				outportb(0x30c,0x00); /* Sinal de partida */
				for(a=0x10;a==0x10;) /* Espera pelo fim de conversao */
					{
						a=inportb(0x305);
						a=a & 0x10;
					}
				a=inportb(0x305);
				b=inportb(0x304);
				a=b+a*256;
				v[m]=(4.*a/4095-2);
				}
			}
		salva3();
	}

salva3()
	{
		FILE *outfile;
		if((outfile = fopen("tensao_j.c","wb")) == NULL)
			{
				perror("Erro em criar tensao_j.c");
				exit(1);
			}
		for (m=0;m<NP;m++)
			{
				fprintf(outfile,"%f\n",v[m]);
			}
		fclose(outfile);
	}



/**********************/
/* Algoritmo Genetico */
/**********************/


genetico()
```

```
{

        x=0;
        erro1=20000;
        /* srand(cont1); */
        /* cont1=cont1+1; */
        for (rod=0;rod<LINHA/2;rod++) /* para completar a matriz corrente */
                {
                        for (cont6=0;cont6<3;cont6++)
                                {
                                        x=random(LINHA);
                                        erro=rk[x];
                                        if (erro<erro1)
                                                {
                                                        erro1=erro;
                                                        x1=x;
                                                }
                                }
                        for (cont7=0;cont7<128;cont7++)
                                {
                                        rec1[cont7]=ini[x1][cont7];    /*    copia    o
string sorteado */
                                        rec2[cont7]=ini[x1][cont7];
                                }

                        for (cont6=0;cont6<3;cont6++)
                                {
                                        x=random(LINHA);
                                        erro=rk[x];
                                        if (x==x1)
                                                {
                                                        x=random(LINHA);
                                                        erro=rk[x];
                                                }
                                        if (erro<erro1)
                                                {
                                                        erro1=erro;
                                                        x1=x;
                                                }
                                }
                        for (cont7=0;cont7<128;cont7++)
                                {
                                        rec3[cont7]=ini[x1][cont7];
                                        rec4[cont7]=ini[x1][cont7];
                                }

                        /* Algoritomo Genetico */

/* 1a */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
```

```
                        {
                        /*      laco1: */
                                x=random(7);
                                if (x>0 && x<8)
                                        {
                                for (p1=0;p1<x;p1++) /* recombinacao */
                                        {
                                                rec1[p1]=rec3[p1];
                                                rec4[p1]=rec2[p1];
                                        }
                                        }
                                /*      if (x>0 && x<64) goto laco1; */
                        }
                        x3=random(100); /* taxa de mutação de 2% */
                if (x3<MUTA)
                        {
                                for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                        {
                                        /*      laco2: */
                                                x=random(7);
                                                if (x>0 && x<8)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                        }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                        }
/* 2a */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                        {
                        /*      laco1: */
                                x=random(7)+8;
                                if (x>7 && x<16)
                                        {
```

```
                                        for (p1=8;p1<x;p1++) /* recombinacao */
                                                {
                                                        rec1[p1]=rec3[p1];
                                                        rec4[p1]=rec2[p1];
                                                }
                                        }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                        {
                                        /*      laco2: */
                                                x=random(7)+8;
                                                if (x>7 && x<16)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                                        }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                }

/* 3a */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+16;
                                        if (x>15 && x<24)
                                                {
                                                for (p1=16;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
```

```
                                                }
                                        }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                        {
                                        /*      laco2: */
                                                x=random(7)+16;
                                                if (x>15 && x<24)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                                        }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                }
/* 4a */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+24;
                                        if (x>23 && x<32)
                                                {
                                                for (p1=24;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
```

```
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+24;
                                                        if (x>23 && x<32)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                                                }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                }

/* 1c */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+64;
                                        if (x>63 && x<72)
                                                {
                                                for (p1=64;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
```

```
                                        {
                                        /*      laco2: */
                                                x=random(7)+64;
                                                if (x>63 && x<72)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                                        }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 2c */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+72;
                                        if (x>71 && x<80)
                                                {
                                                for (p1=72;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+72;
                                                        if (x>71 && x<80)
```

```
                                                {
                                                        if (rec1[x]==0)
                                                                {
                                                                rec1[x]=1;
                                                                }
                                                        else
                                                                {
                                                                rec1[x]=0;
                                                                }
                                                        if (rec4[x]==0)
                                                                {
                                                                rec4[x]=1;
                                                                }
                                                        else
                                                                {
                                                                rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 3c */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+80;
                                        if (x>79 && x<88)
                                                {
                                                for (p1=80;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+80;
                                                        if (x>79 && x<88)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
```

```
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec4[x]=0;
                                                                        }
                                                        }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                }
/* 4c */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+88;
                                        if (x>87 && x<96)
                                                {
                                                for (p1=88;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+88;
                                                        if (x>87 && x<96)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                        }
```

```
                                                        if (rec4[x]==0)
                                                                {
                                                                rec4[x]=1;
                                                                }
                                                        else
                                                                {
                                                                rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 1d */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+96;
                                        if (x>95 && x<104)
                                                {
                                                for (p1=96;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+96;
                                                        if (x>95 && x<104)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {
                                                                        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
                                                                        rec1[x]=0;
                                                                }
                                                                if (rec4[x]==0)
                                                                        {
                                                                        rec4[x]=1;
                                                                        }
```

```
                                                    else
                                                            {
                                                            rec4[x]=0;
                                                            }
                                                    }
                                            /*      if (x>31 && x<64) goto laco2; */
                                            }
                                    }

/* 2d */        x2=random(100); /* Taxa de recombinacao de 90% */
                    if (x2<RECOMB)
                            {
                            /*      laco1: */
                                    x=random(7)+104;
                                    if (x>103 && x<112)
                                            {
                                            for  (p1=104;p1<x;p1++)  /*  recombinacao
*/
                                                    {
                                                            rec1[p1]=rec3[p1];
                                                            rec4[p1]=rec2[p1];
                                                    }
                                            }
                            /*      if (x>0 && x<64) goto laco1; */
                            }
                    x3=random(100); /* taxa de mutação de 2% */
                    if (x3<MUTA)
                            {
                                    for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                            {
                                            /*      laco2: */
                                                    x=random(7)+104;
                                                    if (x>103 && x<112)
                                                            {
                                                            if (rec1[x]==0)
                                                                    {
                                                                    rec1[x]=1;
                                                                    }
                                                            else
                                                                    {
                                                                    rec1[x]=0;
                                                                    }
                                                            if (rec4[x]==0)
                                                                    {
                                                                    rec4[x]=1;
                                                                    }
                                                            else
                                                                    {
                                                                    rec4[x]=0;
```

```
                                                                }
                                                        }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                        }

/* 3d */        x2=random(100); /* Taxa de recombinacao de 90% */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+112;
                                        if (x>111 && x<120)
                                                {
                                                for (p1=112;p1<x;p1++) /* recombinacao
*/                                                                      {
                                                                        rec1[p1]=rec3[p1];
                                                                        rec4[p1]=rec2[p1];
                                                                }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+112;
                                                        if (x>111 && x<120)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {
                                                                                rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {
                                                                                rec1[x]=0;
                                                                                }
                                                                        if (rec4[x]==0)
                                                                                {
                                                                                rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {
                                                                                rec4[x]=0;
                                                                                }
                                                                }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
```

```
                    }

/* 4d */        x2=random(100); /* Taxa de recombinacao de 90% */
                    if (x2<RECOMB)
                        {
                        /*      laco1: */
                                x=random(7)+120;
                                if (x>119 && x<128)
                                    {
                                    for  (p1=120;p1<x;p1++)  /*  recombinacao
*/
                                                {
                                                    rec1[p1]=rec3[p1];
                                                    rec4[p1]=rec2[p1];
                                                }
                                    }
                        /*      if (x>0 && x<64) goto laco1; */
                        }
                    x3=random(100); /* taxa de mutação de 2% */
                    if (x3<MUTA)
                        {
                            for (cont7=0;cont7<NM;cont7++)  /* 3 pontos de
mutacao */
                                {
                                /*      laco2: */
                                    x=random(7)+120;
                                    if (x>119 && x<128)
                                        {
                                            if (rec1[x]==0)
                                                {
                                                rec1[x]=1;
                                                }
                                            else
                                                {
                                                rec1[x]=0;
                                                }
                                            if (rec4[x]==0)
                                                {
                                                rec4[x]=1;
                                                }
                                            else
                                                {
                                                rec4[x]=0;
                                                }
                                        }
                                /*      if (x>31 && x<64) goto laco2; */
                                }
                        }

                    cont8=cont8+2;
```

```
                        cont9=cont9+2;
                        for (cont6=0;cont6<128;cont6++) /* joga os strings modificado
*/
                                {                       /* pelo A.G na matriz corrente */
                                        ini[cont8][cont6]=rec1[cont6];
                                        ini[cont9][cont6]=rec4[cont6];
                                }

                } /* fim das LINHA/2 interacoes e fim da montagem da pop. corrente */

                cont8=-2;
                cont9=-1;

                envio2();

} /* fim da subrotina genetico */


/*************************************/
/* Envio dos strings da pop. corrente */
/*************************************/

envio2()
{
        for (cont11=0;cont11<LINHA;cont11++)
                {
                        for (cont10=0;cont10<128;cont10++)
                                {
                                        h[cont10]=ini[cont11][cont10];
                                }

                        printf("\nString de 128 bits do A.G:\n");
                        for (y=0;y<128;y++)
                                {
                                        printf("%d",h[y]);  /* imprime os strings correto */
                                }
                        printf("\n");

                        envio3();

                        soma=0;
                        for (dd=0;dd<NP;dd++)
                                {
                                        dif[dd]=s[dd]-v[dd];
                                        dif[dd]=pow(dif[dd],2.0);
                                        soma=soma+dif[dd];
                                }
                rk[cont11]=soma;
                }
        incumbente();
```

```
        cont14=cont14+1;
        if (cont14==INTER+1)
                exit(1);

        genetico();

} /* fim da subrotina envio2 */


/*****************************/
/* Envio do string para o CI */
/*****************************/

envio3()
{

reset=1;
clk_m=0;

/* Leitura dos strings da matriz de 16 em 16 bits */

tt=-16;
f=0;
while (f<128)
        {
                tt=tt+16;
                f=f+16;
                p=-1;

                for (c=tt;c<f;c++)
                        {
                                p=p+1;
                                caracter[p]=h[c];
                        }
                for (jj=0;jj<16;jj++)
                        {
                                /* clock em baixa */
                                clk = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                        /*      delay(DELAY); */ /* tempo de acionamento dos leds */

                                /* clock em alta */
                                clk = 1;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                                outportb(0x378,saida);
                        /*      delay(DELAY); */
```

```
                        /* clock em baixa */
                        clk = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em alta */
                        clk_m = 1;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em baixa */
                        clk_m = 0;

        saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY);  */

                        /* enviar 02 */
                        caracter[jj]=caracter[jj]>>1;
                        outportb(0x378,saida);
                /*      delay(DELAY); */
                        }
        }
serra();
}
```

II) Linearização de Saída do VCO

```
/* Circuito integrado de uma Rede de Base Neural Radial */
/* Utiliza 3 Gaussianas */
/* Bit 1 - Dados, bit 2 - Clock, Bit 3 - Reset e bit 4 - Clock master */
/* o Chip , resetado quando o reset , igual ao nivel baixo */

/* Este programa faz o treinamento da Linearização de saída do VCO  */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
```

```
#include <dos.h>

#define DELAY 1
#define AMP 1.5
#define NM 3
#define LINHA 50 /* 73 numero da populacao */
#define NP 300 /* 600 tamanho do vetor */
#define INTER 300 /* 200 numero de interacoes */
#define RECOMB 90 /* taxa de recombinacao 90% */
#define MUTA 20 /* taxa de mutacao 2% */

/* variaveis do vetor handomico */

int y;        /* y(colunas) */
int h[128];   /* vetor tamanho 128 */

/* variaveis dente de serra */

int a,b,m,i,j;
float vco[NP]; /* Vetor tensao */

/* variaveis do vetor reta linear ideal */

int cont2,kk;
float linear[NP];

/* variaveis de acionamento */

int reset;
int clk_m=0;
int clk;
int p,jj,tt,f,c,saida;
int caracter[16];

/* variaveis de leitura e abrupto */

int m,m1,m2,cont15,contf; /* range, comprimento da fundamental */
float vetf[NP],u[NP];
float z;
float fr;

/* variaveis compara */

int cont3,cont4,dd,yy;
float soma;
float dif[NP]; /* vetor diferenca entre a senoide e o norm */
float rk[LINHA]; /* vetor q guarda o erro quadratico de cada string */
int ini[LINHA][128]; /* matriz populacao inicial */

/* variaveis incumbente */
```

```
int cont13;
int best[128]; /* melhor string */
float inc,incumb;

/* variaveis genetico */

int rec1[128],rec2[128],rec3[128],rec4[128];
int p1,x,x1,x2,x3,rod,cont6,cont7,cont8,cont9;
float erro,erro1,inc,incumb;

/* variaveis envio2 */

int cont10,cont11,cont12,cont14;

/*******************************/
/*   PROGRAMA PRINCIPAL        */
/*******************************/

void main()
{
clrscr();

/* para resetar o CI logo no inicio do treinamento */
/* isso eh feito apenas 1 vez */

reset = 0;
saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[16]&0x01);  /* Deslocamento dos
bits */
outportb(0x378,saida);
printf("Sistema Resetado!!!\n\n");
printf("Tecle algo para iniciar o treinamento da Rede!!!");
getch();

inc=10e8;
/* cont1=0; */
cont3=0;
cont4=-1;
cont8=-2; /* linha par da matriz corrente */
cont9=-1; /* linha impar da matriz corrente */
cont11=0; /* linha da matriz corrente */
/* cont12=0; /* coluna do vetor rk da segunda geracao para frente */
cont14=0; /* contador de parada das interacoes */

clrscr();

string();  /* chama a subrotina geracao de string 128 bits aleatorios */
envio();  /* envia os strings para o CI, gera a rampa e faz a aquisicao do sinal de saida
do CI */
/* compara(); */  /* compara os sinais e define o fitnes da populacao inicial */
```

```
} /* Fim do Programa Principal */


/**********************************/
/* Subrotina string 128 bits      */
/**********************************/

string()
{
for (y=0;y<128;y++) /* varre 128 colunas */
        {
                h[y]=random(2);  /* cria a vetor randomico */
        }
for (y=8;y<16;y++) /* iguala os 8 primeiros bits com os 8 segundos bits */
        {
                h[y]=h[y-8];
        }
for (y=72;y<80;y++)
        {
                h[y]=h[y-8];
        }
for (y=104;y<112;y++)
        {
                h[y]=h[y-8];
        }
printf("\nString de 128 bits gerado aleatoriamente:\n");
for (y=0;y<128;y++)
        {
                printf("%d",h[y]);  /* imprime os strings correto */
        }
printf("\n");
}

/*****************************/
/* Envio do string para o CI */
/*****************************/

envio()
{

reset=1;
clk_m=0;

/* Leitura dos strings da matriz de 16 em 16 bits */

tt=-16;
f=0;
while (f<128)
        {
```

```
        tt=tt+16;
        f=f+16;
        p=-1;

        for (c=tt;c<f;c++)
                {
                        p=p+1;
                        caracter[p]=h[c];
                }
        for (jj=0;jj<16;jj++)
                {
                        /* clock em baixa */
                        clk = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */ /* tempo de acionamento dos leds */

                        /* clock em alta */
                        clk = 1;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock em baixa */
                        clk = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em alta */
                        clk_m = 1;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em baixa */
                        clk_m = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* enviar 02 */
                        caracter[jj]=caracter[jj]>>1;
                        outportb(0x378,saida);
                /*      delay(DELAY); */
```

```
                        }
        }
serra();
reta();
}


/*****************************/
/* Subrotina Dente de Serra */
/*****************************/

serra()
        {
                outportb(0x30A,0x00); /* selecao entrada analogica */
                contf=0;
                for(j=0;j<=0x0f;j++) /* geracao dos 12 bits da placa de aquisicao */
                        {
                                for (i=0;i<=0xff;i=(4095./NP)+i)
                                        {
                        for (m=0;m<NP;m++) /* tamanho do vetor vco */
                {
                outportb(0x304,i);
                outportb(0x305,j);
                outportb(0x30c,0x00); /* Sinal de partida */
                for(a=0x10;a==0x10;) /* Espera pelo fim de conversao */
                        {
                                a=inportb(0x305);
                                a=a & 0x10;
                        }
                a=inportb(0x305);
                b=inportb(0x304);
                a=b+a*256;
                vco[m]=(4.*a/4095-2); /* vetor vco */
                }
        freq();
                }
        }
        salva();
        }

salva()
        {
                FILE *outfile;
                if((outfile = fopen("vco.c","wb")) == NULL)
                        {
                                perror("Erro em criar vco.c");
                                exit(1);
                        }
                for (m=0;m<NP;m++)
                        {
```

```
                        fprintf(outfile,"%f\n",vco[m]);
                }
        fclose(outfile);
}

/************************/
/* Subrotina Frequencia */
/************************/

freq()
{
        z=0;
        cont15=0;
        for (m=0;m<NP;m++)
                {
                        z=vco[m+1]-vco[m];
                        if (fabs(z)>AMP) /* dif. dos pontos maior q 1.5 */
                                break;
                }
        for (m1=0;m1<NP;m1++)
                {
                        u[m1]=vco[m+1];
                        m=m+1;
                }
        z=0;
        for (m=0;m<NP;m++)
                {
                        z=u[m+1]-u[m];
                        m2=m;
                        if (fabs(z)>AMP)
                                break;
                }
        for (m1=0;m1<NP;m1++)
                {
                        u[m1]=u[m+1];
                        m=m+1;
                }
        z=0;
        for (m=0;m<NP;m++)
                {
                        z=u[m+1]-u[m];
                        if (fabs(z)>AMP)
                                break;
                }
        m1=m+m2; /* n de pontos equivalentes ao periodo da onda */
        cont15=m1*2;
        fr=(1000.*cont15)/32;
        vetf[contf]=fr;
        contf=contf+1;
        z=0;
```

```
}

/***********************/
/* Funcao Linear Ideal */
/***********************/

reta()
        {
                cont2=0;
                for (kk=0;kk<NP;kk++) /* range pontos */
                        {
                                linear[kk]=cont2+1000;
                                cont2=cont2+(1400/NP);    /*    1000+1400=2400    eh    a
frequencia maxima */
                        }
        salva1();
        compara();
        }

salva1()
        {
                FILE *outfile;
                if((outfile = fopen("linear.c","wb")) == NULL)
                        {
                                perror("Erro em criar linear.c");
                                exit(1);
                        }
                for (kk=0;kk<NP;kk++)
                        {
                                fprintf(outfile,"%f\n",linear[kk]);
                        }
                fclose(outfile);
        }

/*************************************************/
/* Compara o sinal de saida com uma senoide ideal */
/*************************************************/

compara()
                {
                        soma=0;
                        for (dd=0;dd<NP;dd++)
                                {
                                        dif[dd]=linear[dd]-vetf[dd];
                                        dif[dd]=pow(dif[dd],2.0);
                                        soma=soma+dif[dd];
                                }
                        rk[cont3]=soma;
                /*      contf=0; */
```

```
                        cont3=cont3+1;
                        cont4=cont4+1;
                        for (yy=0;yy<128;yy++)
                                {
                                        ini[cont4][yy]=h[yy];
                                }
                if (cont4==LINHA-1)
                        {
                                incumbente();
                                genetico();
                        }
                else
                        {
                                string();
                                envio();
                        }
                } /* fim da subrotina compara */


/**************/
/* Incumbente */
/**************/

incumbente()
        {

                for (cont6=0;cont6<LINHA;cont6++)
                        {
                                incumb=rk[cont6];
                                if (incumb<inc)
                                        {
                                                inc=incumb;   /* incumbente */
                                                for (cont13=0;cont13<128;cont13++)
                                                        {

        best[cont13]=ini[cont6][cont13];
                                                        }
                                        salva2();
                                        salva4();
                                        serra2();
                                        }
                        }
}

salva2()
        {
                FILE *outfile;
                if((outfile = fopen("best.c","wb")) == NULL)
                        {
                                perror("Erro em criar best.c");
```

```
                exit(1);
            }
        for (cont2=0;cont2<128;cont2++)
            {
                fprintf(outfile,"%d\n",best[cont2]);
            }
        fclose(outfile);
    }

salva4()
{
    FILE *outfile;
    if((outfile = fopen("inc.c","wb")) == NULL)
        {
            perror("Erro em criar inc.c");
            exit(1);
        }
        fprintf(outfile,"%f",inc);
        fclose(outfile);
}


serra2()
    {
        outportb(0x30A,0x00); /* selecao entrada analogica */
        contf=0;
        for(j=0;j<=0x0f;j++) /* geracao dos 12 bits da placa de aquisicao */
            {
            for (i=0;i<=0xff;i=(4095./NP)+i)
            {
            for (m=0;m<NP;m++) /* tamanho do vetor vco */
            {
                outportb(0x304,i);
                outportb(0x305,j);
                outportb(0x30c,0x00); /* Sinal de partida */
                for(a=0x10;a==0x10;) /* Espera pelo fim de conversao */
                    {
                        a=inportb(0x305);
                        a=a & 0x10;
                    }
                a=inportb(0x305);
                b=inportb(0x304);
                a=b+a*256;
                vco[m]=(4.*a/4095-2); /* vetor vco */
            }
            freq();
            }
    }
    salva3();
    }
```

```
salva3()
	{
		FILE *outfile;
		if((outfile = fopen("vcofbest.c","wb")) == NULL)
			{
				perror("Erro em criar vcofbest.c");
				exit(1);
			}
		for (m=0;m<NP;m++)
			{
				fprintf(outfile,"%f\n",vetf[m]);
			}
		fclose(outfile);
	}

/**********************/
/* Algoritmo Genetico */
/**********************/

genetico()
{
x=0;
erro1=10e8;
for (rod=0;rod<LINHA/2;rod++) /* para completar a matriz corrente */
	{
		for (cont6=0;cont6<3;cont6++)
			{
				x=random(LINHA);
				erro=rk[x];
				if (erro<erro1)
					{
						erro1=erro;
						x1=x;
					}
			}
		for (cont7=0;cont7<128;cont7++)
			{
				rec1[cont7]=ini[x1][cont7]; /* copia o string sorteado */
				rec2[cont7]=ini[x1][cont7];
			}

		for (cont6=0;cont6<3;cont6++)
			{
				x=random(LINHA);
				erro=rk[x];
				if (x==x1)
			{
		x=random(LINHA);
		erro=rk[x];
```

```
        }
if (erro<erro1)
        {
                erro1=erro;
                x1=x;
        }
}
        for (cont7=0;cont7<128;cont7++)
                {
                        rec3[cont7]=ini[x1][cont7];
                        rec4[cont7]=ini[x1][cont7];
                }

/* Algoritomo Genetico */

/* 1a */        x2=random(100); /* Taxa de recombinação  */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7);
                                        if (x>0 && x<8)
                                                {
                                                for (p1=0;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7);
                                                        if (x>0 && x<8)
                                                                {
                                                                        if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
                                                                                }
```

```
                                                        if (rec4[x]==0)
                                                                {

        rec4[x]=1;
                                                                }
                                                        else
                                                                {

        rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 2a */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+8;
                                        if (x>7 && x<16)
                                                {
                                                for (p1=8;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* NM pontos
de mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+8;
                                                if (x>7 && x<16)
                                                        {
                                                                if (rec1[x]==0)
                                                                {

        rec1[x]=1;
                                                                }
                                                        else
                                                                {

        rec1[x]=0;
                                                                }
```

```
                                                        if (rec4[x]==0)
                                                                {

        rec4[x]=1;
                                                                }
                                                        else
                                                                {

        rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 3a */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+16;
                                        if (x>15 && x<24)
                                                {
                                                for (p1=16;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+16;
                                                        if (x>15 && x<24)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec1[x]=0;
                                                                        }
```

```
                                                        if (rec4[x]==0)
                                                                {

        rec4[x]=1;
                                                                }
                                                        else
                                                                {

        rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 4a */        x2=random(100); /* Taxa de recombinção */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+24;
                                        if (x>23 && x<32)
                                                {
                                                for (p1=24;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                        /*      if (x>0 && x<64) goto laco1; */
                                        }
                                x3=random(100); /* taxa de mutação */
                                if (x3<MUTA)
                                        {
                                                for (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+24;
                                                        if (x>23 && x<32)
                                                        {
                                                                if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec1[x]=0;
                                                                        }
```

```
                                                                        if (rec4[x]==0)
                                                                                {

        rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec4[x]=0;
                                                                                }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                        }

/* 1c */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+64;
                                        if (x>63 && x<72)
                                                {
                                                for (p1=64;p1<x;p1++) /* recombinacao */
                                                                {
                                                                        rec1[p1]=rec3[p1];
                                                                        rec4[p1]=rec2[p1];
                                                                }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+64;
                                                        if (x>63 && x<72)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
                                                                                }
```

```
                                                        if (rec4[x]==0)
                                                                {

        rec4[x]=1;
                                                                }
                                                        else
                                                                {

        rec4[x]=0;
                                                                }
                                                }
                                        /*      if (x>31 && x<64) goto laco2; */
                                        }
                                }

/* 2c */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+72;
                                        if (x>71 && x<80)
                                                {
                                                for (p1=72;p1<x;p1++) /* recombinação */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                        /*      if (x>0 && x<64) goto laco1; */
                                        }
                                x3=random(100); /* taxa de mutação */
                                if (x3<MUTA)
                                        {
                                                for (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutaçã */
                                                {
                                                /*      laco2: */
                                                x=random(7)+72;
                                                        if (x>71 && x<80)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec1[x]=0;
                                                                        }
```

```
                                                                        if (rec4[x]==0)
                                                                                {

        rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec4[x]=0;
                                                                                }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                        }

/* 3c */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+80;
                                        if (x>79 && x<88)
                                                {
                                                for (p1=80;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* NM pontos
de mutacao */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+80;
                                                        if (x>79 && x<88)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
                                                                                }
```

```
                                                                        if (rec4[x]==0)
                                                                                {

        rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec4[x]=0;
                                                                                }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                        }

/* 4c */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+88;
                                        if (x>87 && x<96)
                                                {
                                                for (p1=88;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                                if (x3<MUTA)
                                        {
                                                for (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+88;
                                                        if (x>87 && x<96)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
                                                                                }
```

```
                                                                        if (rec4[x]==0)
                                                                                {

        rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec4[x]=0;
                                                                                }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                        }

/* 1d */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+96;
                                        if (x>95 && x<104)
                                                {
                                                for (p1=96;p1<x;p1++) /* recombinacao */
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação de 2% */
                        if (x3<MUTA)
                                {
                                        for (cont7=0;cont7<NM;cont7++)  /* NM pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+96;
                                                        if (x>95 && x<104)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
                                                                                }
```

```
                                                                        if (rec4[x]==0)
                                                                                {

        rec4[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec4[x]=0;
                                                                                }
                                                                }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                        }

/* 2d */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+104;
                                        if (x>103 && x<112)
                                                {
                                                for  (p1=104;p1<x;p1++)  /*  recombinacao
*/
                                                                {
                                                                        rec1[p1]=rec3[p1];
                                                                        rec4[p1]=rec2[p1];
                                                                }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                        if (x3<MUTA)
                                {
                                        for  (cont7=0;cont7<NM;cont7++)   /*  NM  pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+104;
                                                        if (x>103 && x<112)
                                                                {
                                                                        if (rec1[x]==0)
                                                                                {

        rec1[x]=1;
                                                                                }
                                                                        else
                                                                                {

        rec1[x]=0;
```

```
                                                                }
                                                                if (rec4[x]==0)
                                                                        {

        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec4[x]=0;
                                                                        }
                                                        }
                                                /*      if (x>31 && x<64) goto laco2; */
                                                }
                                        }

/* 3d */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+112;
                                        if (x>111 && x<120)
                                                {
                                                for  (p1=112;p1<x;p1++)  /*  recombinacao
*/
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                        if (x3<MUTA)
                                {
                                        for  (cont7=0;cont7<NM;cont7++)   /* NM pontos
de mutação */                                                   {
                                                /*      laco2: */
                                                        x=random(7)+112;
                                                        if (x>111 && x<120)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec1[x]=0;
```

```
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {

        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec4[x]=0;
                                                                        }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                        }

/* 4d */        x2=random(100); /* Taxa de recombinação */
                        if (x2<RECOMB)
                                {
                                /*      laco1: */
                                        x=random(7)+120;
                                        if (x>119 && x<128)
                                                {
                                                for  (p1=120;p1<x;p1++)  /*  recombinacao
*/
                                                        {
                                                                rec1[p1]=rec3[p1];
                                                                rec4[p1]=rec2[p1];
                                                        }
                                                }
                                /*      if (x>0 && x<64) goto laco1; */
                                }
                        x3=random(100); /* taxa de mutação */
                        if (x3<MUTA)
                                {
                                        for  (cont7=0;cont7<NM;cont7++)   /*  NM  pontos
de mutação */
                                                {
                                                /*      laco2: */
                                                        x=random(7)+120;
                                                        if (x>119 && x<128)
                                                                {
                                                                if (rec1[x]==0)
                                                                        {

        rec1[x]=1;
                                                                        }
                                                                else
                                                                        {
```

```
        rec1[x]=0;
                                                                        }
                                                                if (rec4[x]==0)
                                                                        {

        rec4[x]=1;
                                                                        }
                                                                else
                                                                        {

        rec4[x]=0;
                                                                        }
                                                                }
                                                        /*      if (x>31 && x<64) goto laco2; */
                                                        }
                                }


                        cont8=cont8+2;
                        cont9=cont9+2;
                        for (cont6=0;cont6<128;cont6++) /* joga os strings modificado
*/
                                {                       /* pelo A.G na matriz corrente */
                                        ini[cont8][cont6]=rec1[cont6];
                                        ini[cont9][cont6]=rec4[cont6];
                                }

                } /* fim das LINHA/2 interações e fim da montagem da pop. corrente */

        cont8=-2;
        cont9=-1;

        envio2();

} /* fim da subrotina genético */


/*************************************/
/* Envio dos strings da pop. Corrente */
/*************************************/

envio2()
{
        for (cont11=0;cont11<LINHA;cont11++)
                {
                        for (cont10=0;cont10<128;cont10++)
                                {
                                        h[cont10]=ini[cont11][cont10];
                                }
```

```
                        printf("\nString de 128 bits do A.G:\n");
                        for (y=0;y<128;y++)
                                {
                                        printf("%d",h[y]);  /* imprime os strings correto */
                                }
                        printf("\n");

                        envio3();

                        contf=0;
                        soma=0;
                        for (dd=0;dd<NP;dd++)
                                {
                                        dif[dd]=linear[dd]-vetf[dd];
                                        dif[dd]=pow(dif[dd],2.0);
                                        soma=soma+dif[dd];
                                }
                rk[cont11]=soma;
        }
        incumbente();

        cont14=cont14+1;
        if (cont14==INTER+1)
                exit(1);
        genetico();

} /* fim da subrotina envio2 */


/*****************************/
/* Envio do string para o CI */
/*****************************/

envio3()
{
reset=1;
clk_m=0;
/* Leitura dos strings da matriz de 16 em 16 bits */

tt=-16;
f=0;
while (f<128)
        {
                tt=tt+16;
                f=f+16;
                p=-1;

                for (c=tt;c<f;c++)
                        {
```

```
                        p=p+1;
                        caracter[p]=h[c];
                }
        for (jj=0;jj<16;jj++)
                {
                        /* clock em baixa */
                        clk = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */ /* tempo de acionamento dos leds */

                        /* clock em alta */
                        clk = 1;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock em baixa */
                        clk = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em alta */
                        clk_m = 1;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY); */

                        /* clock master em baixa */
                        clk_m = 0;

saida=(reset<<2)+(clk_m<<3)+(clk<<1)+(caracter[jj]&0x01);
                        outportb(0x378,saida);
                /*      delay(DELAY);  */

                        /* enviar 02 */
                        caracter[jj]=caracter[jj]>>1;
                        outportb(0x378,saida);
                /*      delay(DELAY); */
                }
        }
serra();
}
```