UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

# "Uma Ferramenta Alternativa para Síntese de Circuitos Lógicos Usando a Técnica de Circuito Evolutivo"

**EDILTON FURQUIM GOULART SOBRINHO**

**Orientador**: Profa. Dra. Suely Cunha Amaro Mantovani

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

Ilha Solteira – SP
junho/2007

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação - Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

Goulart Sobrinho, Edilton Furquim
G694f Uma ferramenta alternativa para síntese de circuitos lógicos usando a técnica de circuito evolutivo / Edilton Furquim Goulart Sobrinho. -- Ilha Solteira : [s.n.], 2007
82 p. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2007

Orientador: Suely Cunha Amaro Mantovani
Bibliografia: p. 75-78

1. Circuitos integrados digitais. 2. Algoritmos genéticos. 3. Circuitos lógicos. 4. Máquinas-ferramenta - Projetos. 5. Hardware evolutivo.

*Dedico esse trabalho aos meus pais, Júlio Furquim Goulart e Dalva Queiroz Oliveira Furquim, aos meus irmãos Ana Júlia Queiroz Furquim e Júlio Furquim Goulart Junior, ao meu sobrinho Magno Queiroz Furquim Moraes e minha companheira amada Giselly de Oliveira Lima. Agradeço-lhes pelo apoio, dedicação e carinho recebidos.*

# AGRADECIMENTOS

Não seria possível realizar esta dissertação sem o apoio, influência e encorajamento de um grande número de pessoas, por esses e diversos outros motivos gostaria de agradecer:

- ✓ À minha orientadora Suely Cunha Amaro Mantovani que direcionou os meus estudos sempre se mostrando disposta a elucidar minhas dúvidas e principalmente por apoiar-me juntamente com seu esposo José Roberto Sanches Mantovani no momento em que mais precisei.
- ✓ Ao professor Nobuo Oki que participou do exame de qualificação e da banca de defesa do mestrado com importantes contribuições enriquecendo o trabalho desenvolvido.
- ✓ Ao professor José Raimundo de Oliveira que fez parte da banca avaliadora na defesa e abrilhantou aquele momento com pensamentos, idéias e palavras até hoje presentes em meus pensamentos.
- ✓ Ao professor Rubén Augusto Romero Lázaro pelos ensinamentos repassados e pelas contribuições para o desenvolvimento da pesquisa realizada.
- ✓ Aos professores e amigos Carlos Roberto Minussi, Dalgerti Lelis Milanese, Dílson Paschoarelli Junior e Sergio Kurokawa pelos ensinamentos, apoio, companheirismo e palavras que só quem tem a bagagem da vida poderia repassar.
- ✓ Ao professor Marcelo Carvalho Minhoto Teixeira pela prontidão em atender minhas dúvidas no que diz respeito aos trâmites burocráticos se mostrando sempre prestativo e presente.
- ✓ A todos os docentes, funcionários e estagiários do Departamento de Engenharia Elétrica, e em especial ao Deoclécio Mitsuiti Kosaka, Luzinete Maria de Oliveira e José Roberto Campos.
- ✓ Aos meus colegas de trabalho Andréa Protto, Alfredo Bonini Neto, Célia Regina, Alessandra Bonato Altran, Carlos Alberto Febres Tapia, Flavio Faria, Ápio Carniello, Hélio Clementino dos Santos, Sérgio Nazário, Renato Cardoso dos Santos, Mariza Akiko Utida, Silvia Lopes de Sena Taglialenha, Fabiano Alves de Souza, Fernando Ochôa, Renato Nagashima e Josivaldo Godoy.

- ✓ Aos amigos Guilherme e Daniela Melo, André Scagnolato, André e Eliane Pereira, Carlos Eduardo da Silva Santos, Tercio Alberto dos Santos Filho, Edson e Luzia Righeto, Marcos Antonio Estremote, Rodrigo Serra Daltin, Marcelo, Francisco, Mara Lopes e Walter, Jaine Henrique Canossa, Jorge Henrique Depieri Medeiros, Nilton Vieira Junior, Fernado Sanches e Cilene e Celso Tadao Miasaki.
- ✓ Aos funcionários da Unesp João Josué Barbosa, Onilda Naves de Oliveira Akasaki, Adelaide Amaral dos Santos Passipieri e Maria Fátima Sabino, pela educação, destreza e prontabilidade no atendimento.
- ✓ Aos meus pais, Júlio Furquim Goulart e Dalva Queiroz de Oliveira Furquim que nos altos e baixos da minha vida deram-me os suportes financeiro e psicológico necessários para enfrentar diferentes adversidades, e que também proporcionaram a continuidade de meus estudos para a conquista de mais esta etapa.
- ✓ Aos meus irmãos Ana Júlia Queiroz Furquim e Julio Furquim Goulart Junior que partilharam experiências, participaram na construção do meu caráter e fortemente contribuíram para a conclusão deste trabalho.
- ✓ À Maria Tereza de Paula Santos Furquim por me mostrar o homem que é meu irmão e me dar a felicidade de fazer parte da minha família.
- ✓ À minha companheira amada Giselly de Oliveira Lima pela confiança, carinho e apoio fundamentais para a transposição desta etapa e certamente de muitas outras que virão em nossa caminhada juntos.
- ✓ Em especial ao meu sobrinho Magno Queiroz Furquim Moraes que desde a sua vida embrionária mostrou a garra que se deve ter para viver e até hoje mostra a todos a alegria em estar vivo e partilhar do seio desta família.
- ✓ À Capes - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pelo auxílio financeiro.
- ✓ A todos que confiaram em mim e, de alguma forma, contribuíram para que tudo fosse possível.

*“A mente que se abre a uma nova idéia*
*jamais voltará ao seu tamanho original”.*
*Albert Einstein*

# RESUMO

Neste trabalho descreve-se uma metodologia para síntese e otimização de circuitos digitais, usando a teoria de algoritmos evolutivos e como plataforma os dispositivos reconfiguráveis, denominada Hardware Evolutivo do inglês- *Evolvable Hardware* – EHW. O EHW, tornou-se viável com o desenvolvimento em grande escala dos dispositivos reconfiguráveis, *Programmable Logic Devices* (PLD's), cuja arquitetura e função podem ser determinadas por programação. Cada circuito pode ser representado como um indivíduo em um processo evolucionário, evoluindo-o através de operações genéticas para um resultado desejado. Como algoritmo evolutivo, aplicou-se o Algoritmo Genético (AG), uma das técnicas da computação evolutiva que utiliza os conceitos da genética e seleção natural. O processo de síntese aplicado neste trabalho, inicia por uma descrição do comportamento do circuito, através de uma tabela verdade para circuitos combinacionais e a tabela de estados para os circuitos seqüenciais. A técnica aplicada busca o arranjo correto e minimizado do circuito que desempenhe uma função proposta. Com base nesta metodologia, são implementados alguns exemplos em duas diferentes representações (mapas de fusíveis e matriz de portas lógicas).

**Palavras-chave**: *Hardware* Evolutivo, Síntese de Circuitos Digitais, Algoritmo Genético, Circuitos Combinacionais, Máquina de Estados.

# ABSTRACT

In this work was described a methodology for optimization and synthesis of digital circuits, which consist of evolving circuits through evolvable algorithms using as platforms reconfigurable devices, denominated Evolvable Hardware (EHW). It was became viable with the large scale development of reconfigurable devices, whose architecture and function can be determined by programming. Each circuit can be represented as an individual within an evolutionary process, evolving through genetic operations to desire results. Genetic Algorithm (GA) was applied as evolutionary algorithm where this technique evolvable computation as concepts of genetics and natural selection. The synthesis process applied in this work starts from a description from the circuits behavior. Trust table for combinatorial circuits and state transition table for sequential circuits were used for synthesis process. This technic applied search the correct arrange and minimized circuit which response the propose function. Based on this methodology, some examples are implemented in two different representations (fuse maps and logic gate matrices).



**Key-words**: Evolvable Hardware, Synthesis of Digital Circuits, Genetic Algorithm, Combinatorial Circuits, State Machine.

# Lista de Ilustrações

## Sumário

# 1 INTRODUÇÃO

A Computação Evolutiva é uma área de estudo que trabalha com algoritmos inspirados em aspectos essenciais da evolução natural e os aplica na solução de problemas nas mais diversas áreas do conhecimento humano, principalmente, com o propósito de otimização de diferentes aspectos de projetos, através de seus algoritmos inteligentes, como Redes Neurais, Algoritmos Evolutivos, Lógica Fuzzy dentre outros.

Na categoria algoritmos evolutivos (MICHALEWICZ, 1996), apresentam-se vários tipos que recebem denominações diferentes de acordo com certas particularidades: Algoritmo Genético-AG, Programação Genética-PG, Programção Evolutiva-PE , Estratégias Evolutivas-EE, entre outros. Estes algoritmos estão baseados na seleção natural, recombinação de material genético e a mutação.

O Algorigmo Genético tem sido aplicado a uma variedade de problemas, sendo uma importante ferramenta em máquina de aprendizagem e otimização de funções, através do uso de estratégias inteligentes de busca. Baseia-se no **princípio da seleção natural**, de forma que os indivíduos melhores dotados têm maiores chances de sobreviver a um ambiente competitivo. As características específicas definidas pelo conteúdo genético (gene) do indivíduo determinam sua capacidade de sobrevivência. A evolução dos indivíduos dá-se com a utilização dos operadores de recombinação e mutação. O AG tem sido mais utilizado devido sua simplicidade e aplicações relacionados a otimização e síntese de sistemas (TOMASSINI, 1997).

Dentro do contexto de síntese de projeto, neste trabalho utilizam-se os AGs e os dispositivos reconfiguráveis projetados nos anos 80, para compor o conceito de Hardware evolutivo- EHW do inglês *Evolvable Hardware* .

Os circuitos evolutivos ou EHW são projetos de circuito que podem ter sua arquitetura e funções modificadas de maneira dinâmica e autônoma de forma a atender às

mudanças no meio, diferindo dos circuitos tradicionais cuja estrutura e funções são fixadas na fabricação. EHW tem como característica principal, à adaptação de hardware para sistemas como os robôs que apresentam comportamento dinâmico, tornando-se viável com o desenvolvimento em grande escala dos dispositivos reconfiguráveis, FPGA's (*Field Programmable Logic Arrays*), CPLD's (*Complex Programmable Logic Devices*), etc. Os dispositivos reconfiguráveis, podem ter sua arquitetura e função determinadas por algoritmos evolutivos, basta para isso codificar de forma adequada as portas lógicas através de uma representação cromossômica. Portanto, EHW utiliza algoritmos evolutivos e dispositivos lógicos programáveis e vem sendo usado no desenvolvimento de projeto de circuitos analógicos e digitais. Em Zebulum et al. (ZEBULUM et al., 1997) encontram-se os primeiros estudos em síntese de circuitos digitais.

A seguir apresenta-se os objetivos da tese, metodologia utilizada , descrição e organização do trabalho.

## 1.1 OBJETIVOS

O objetivo deste trabalho é a Síntese de Circuitos Digitais, abrangendo circuitos combinacionais e sequenciais usando a teoria de Circuito Evolutivo. Esta terminologia envolve a utilização do AG usado para a otimização e/ou síntese de circuitos (KOZA et al., 1997; LOUIS e RAWLINS, 1991; THOMPSON et al., 1996).

Mostra o desempenho desta técnica em projetos de circuitos digitais e um algoritmo calibrado, que é usado para gerar alguns exemplos de projeto, tendo como conhecimento prévio sua tabela verdade.

Portanto, o objetivo maior é a obtenção de novas ferramentas de projeto de circuitos, competitivas com o estado da arte em eletrônica digital e baseadas unicamente na computação evolutiva. Além disso, busca-se obter métodos de síntese de circuitos que tenham pouca dependência do conhecimento prévio do circuito, diferindo das técnicas usuais de projeto.

## 1.2 METODOLOGIAS EVOLUTIVAS

A metodologia investigada nesta dissertação abrange os mecanismos de seleção, recombinação e mutação, além do uso de uma representação cromossômica com portas lógicas , onde o alvo é a convergência do algoritmo para um circuito correto e otimizado baseado em (COELLO et al., 2001).

## 1.3 DESCRIÇÃO E ORGANIZAÇÃO DO TRABALHO

Esta dissertação apresenta o conceito de circuito evolutivo e investiga sua utilização na síntese de circuitos digitais. Os algoritmos genéticos implementados incluem sistemas com memória genética, usam representação fixa , e a evolução é tratada como uma função de um único objetivo. Sistemas evolutivos com memória genética são realizados através da preservação e re-utilização de material genético.

Estuda-se a evolução dos circuitos digitais por portas lógicas conforme Coello e a idéia da introdução de fios, para a otimização do circuito. Mostra-se ainda a representação por mapa de fusíveis (MANTOVANI e OLIVEIRA, 2004). Essas representações são utilizadas no estudo da síntese de circuitos combinacionais básicos como somadores e decodificadores e em circuitos sequenciais , como detetores de seqüência e contadores .

Para validar as simulações efetuadas foram realizados testes na placa UPX2 – fabricante Altera (ALTERA, 2006) que contém o circuito integrado EPF10K70RC240-4 da família Flex 10K do mesmo fabricante .

Esta dissertação possui além deste, o capítulo 2 que apresenta um breve levantamento sobre o AG e seus operadores. O capítulo 3 que apresenta uma revisão bibliográfica mostrando os caminhos traçados por esta teoria e sua aplicabilidade atual.e descreve a técnica do circuito evolutivo.

As técnicas aplicadas no trabalho fazem parte do capítulo 4. No capítulo 5 apresenta-se os exemplos implementados e os resultados obtidos , simulações e circuitos . Finalmente, no capítulo 6 são apresentas as conclusões e sugestões para futuros trabalhos.

# 2. ALGORITMO GENÉTICO

Neste capítulo descreve-se os algoritmos genéticos, que são procedimentos computacionais empregados na solução de problemas que executam otimização, através de um processo de aprimoramento de um conjunto de soluções, e seus operadores inspirados na evolução biológica natural.

## 2.1 INTRODUÇÃO

Desenvolvidos por John Holland visando criar programas de computador com as características dos mecanismos naturais para aplicar em processos adaptativos, hoje os algoritmos genéticos tem encontrado sua maior fonte de aplicação na área de otimização, particularmente em problemas em que o tamanho e/ou complexidade do espaço de busca torna inviável a utilização de técnicas convencionais.

Na literatura encontram-se três classes de métodos de busca ou otimização, os baseados em cálculo, aqueles baseados em enumeração e os aleatórios. Os métodos que se baseiam em cálculo computam o gradiente da função a ser otimizada. Os métodos de

enumeração, que amostram todos os espaços de busca , são eficientes somente quando usados com técnicas que usam localidade. Os métodos de busca aleatória empregam, em geral, a comparação entre um ponto gerado aleatório e outro gerado previamente, retendo o melhor. Estes métodos representam a melhor solução apenas para problemas em que o espaço de busca é extremamente grande e complexo.

Por outro lado, os Algoritmos Genéticos (AG's) diferem das técnicas descritas acima pelo fato de trabalharem uma população de soluções e de usarem regras de transição probabilísticas. Além disso, não necessitam de nenhuma informação adicional sobre o espaço de busca (emprego de derivadas, por exemplo). Diz-se que os algoritmos genéticos são considerados cegos quando empregados em sua forma mais pura (GOLDBERG, 1989).

Os algoritmos genéticos estão baseados no princípio da seleção natural, onde os indivíduos melhor dotados têm maiores chances de sobreviver a um ambiente competitivo e somente estes repassam suas características aos seus descendentes.

Os algoritmos genéticos, assim como na evolução biológica, se utilizam de alguns operadores como seleção, crossover (cruzamento) e mutação e da aplicação de uma função de avaliação de aptidão. Em cada geração, o algoritmo busca sua evolução visando uma solução ótima para um determinado problema.

Por sua característica robusta, esta técnica de otimização vem sendo empregada em diversas áreas de conhecimento como otimização de problemas numéricos e combinatoriais, escalonamento de tarefas, desenvolvimento de *layouts* (modelos) para circuitos, aprendizagem de máquinas, etc. (ZEBULUM, 1999).

Tipicamente, como no processo de evolução natural, para resolver um problema de otimização, os AG's empregam um conjunto de configurações iniciais (populações de indivíduos) de tamanho constante (ou variável) que representam as soluções candidatas para o problema. Estas configurações ou indivíduos são geralmente codificadas em cadeias ou *strings* binários. Todos os indivíduos da população são classificados pela qualidade de sua correspondente função objetivo ou função de *fitness*.

Cada elemento da população melhor qualificado ou que apresenta alto *fitness* tem maior probabilidade de gerar os elementos da nova população. A nova população é obtida através dos operadores recombinação e mutação. Esse processo termina quando é obedecido o critério de parada que pode ser o melhor indivíduo ou um número fixo de gerações. O algoritmo genético é ilustrado pelo fluxograma apresentado na Figura 2.1 e os detalhes de cada fase no fluxograma são apresentados no restante deste capítulo.

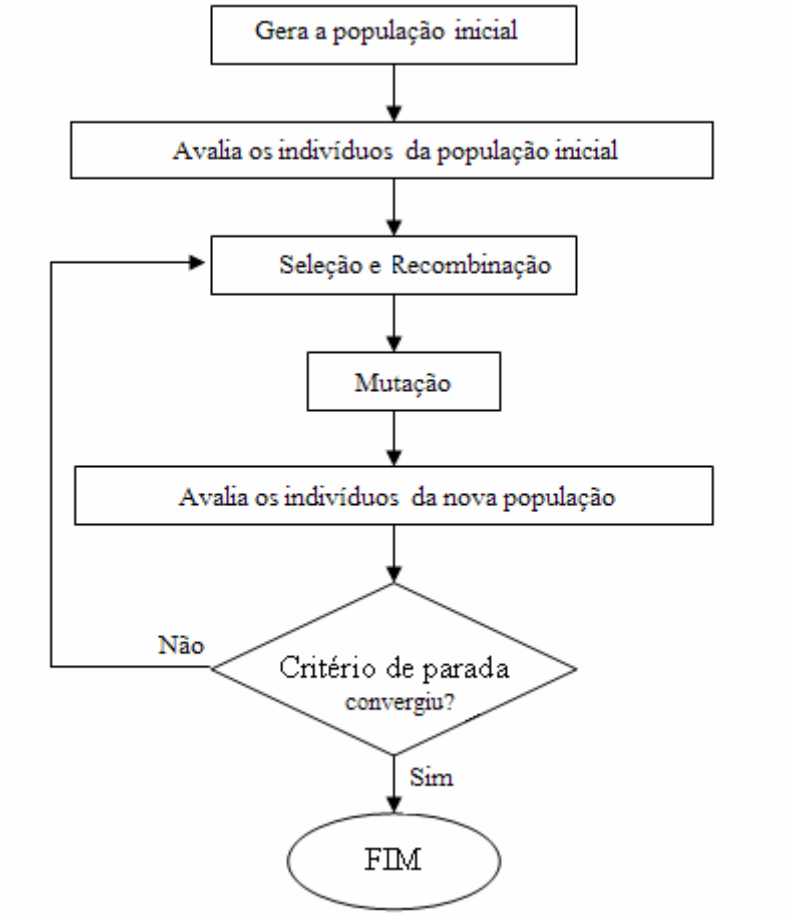



Figura 2.1 – Fluxograma do Algoritmo Genético.

## 2.2 GERAÇÃO DA POPULAÇÃO INICIAL

A população inicial é formada por candidatos à solução do problema podendo ser gerada aleatoriamente ou não, o importante é garantir a diversidade da população e a expansão da área de busca. Algumas experiências (TOMASSINI, 1997) indicam que quando a população inicial é gerada de maneira aleatória há um esforço computacional maior do que, quando se emprega uma população inicial gerada por meios de estratégias baseadas no conhecimento do problema sob estudo. Portanto, o método de geração da população irá depender da complexidade do problema, e da inviabilidade do esforço computacional.

A geração da população envolve a representação do problema ou mapeamento das possíveis soluções presentes no espaço de busca em uma estrutura de dados que possa ser manipulada computacionalmente. Tipicamente, os AG's codificam possíveis soluções em palavras binárias, ou cromossomos em um sistema biológico, de tamanho fixo ou variável. No caso da representação de tamanho fixo frequentemente usada nos algoritmos evolutivos, o tamanho de cromossomo, *Tc* é constante. Sistemas evolutivos com representação variável são aqueles em que o tamanho do cromossomo varia ao longo do processo evolutivo e/ou entre indivíduos de uma população.

Os AG's , na forma como foram inicialmente concebidos, utilizam representação binária de tamanho fixo (GOLDBERG, 1989), ou seja, cada posição ou alelo do cromossomo é representada por um bit, e o número de genes permanece constante ao longo do processo seletivo. Apesar desta definição em representação binária nos AG's, atualmente é bastante usual a utilização da representação inteira, onde cada posição de um cromossomo pode assumir k diferentes números inteiros. A variável *k* é denominada de alfabeto de representação. Portanto, o alfabeto e o tamanho da representação determinam o tamanho do espaço de busca, dado por $k^{Tc}$, onde *Tc* é o tamanho do cromossomo (ZEBULUM, 1999).

Esta pesquisa emprega a representação inteira de tamanho fixo onde o cromossomo (ou genótipo) é visto como uma coleção de genes, onde um gene é constituído por uma ou mais posições do cromossomo. Definindo a variável r como o número de posições do gene e *n* o número de genes do cromossomo, o tamanho do cromossomo, *Tc* é dado por:

$$Tc = n \times r \tag{2.1}$$

## 2.3 FUNÇÃO DE AVALIAÇÃO

A maioria dos problemas de interesse prático deve atender a múltiplos objetivos, uma vez que diversas especificações devem ser levadas em consideração. Porém, quando apenas um critério deve ser atendido, a determinação da função de avaliação de aptidão (função objetivo) é simples. Este critério é representado por uma função objetivo.

Uma função de avaliação resulta em um escalar como medida de desempenho de um determinado indivíduo, portanto são necessários métodos de conversão de uma medida de aptidão vetorial em um escalar.

## 2.4 OPERADOR DE SELEÇÃO

O operador de seleção é um componente essencial de AG's . Um mecanismo de seleção é caracterizado pela pressão seletiva ou intensidade de seleção que o mesmo introduz no algoritmo genético. Dentre os principais mecanismos de seleção encontra-se na literatura a chamada seleção proporcional, por torneios, com truncamento, por normalização linear e por normalização exponencial.

No caso da **seleção proporcional**, a probabilidade de um indivíduo ser selecionado é simplesmente proporcional ao seu valor de aptidão, isto é:

$$p_i = \frac{f_i}{M} N \tag{2.2}$$

onde $p_i$ é a probabilidade de seleção de um indivíduo i, $f_i$ é a aptidão do mesmo, $N$ é o tamanho da população e $M$ é aptidão média da população. Este método de seleção proporcional gera "super" indivíduos e competição próxima. O primeiro ocorre quando um indivíduo apresenta uma aptidão bem maior que a dos restantes, ocasionando uma convergência rápida do algoritmo. O segundo problema ocorre quando indivíduos apresentam aptidões semelhantes, mas não idênticas; neste caso, a intensidade de seleção pode ser bem menor do que a desejável.

Na **seleção por torneio**, um grupo de t indivíduos é aleatoriamente escolhido, e o indivíduo de melhor aptidão é selecionado. Neste caso observa-se por estudos realizados, que a pressão seletiva aumenta, a medida em que o número de indivíduos *t* no torneio, aumenta.

Na avaliação dos cromossomos neste trabalho, definem-se as configurações candidatas a gerar descendentes. Os descendentes são escolhidos realizando-se ***t*** jogos sendo que *t* é igual ao número ***n*** de indivíduos da população. A cada jogo foram escolhidos ***x*** indivíduos e a configuração vencedora dentre esses é aquela que tem a função objetivo de melhor qualidade.

No mecanismo de **seleção por truncamento**, dado um limiar *T*, apenas os *T* melhores indivíduos podem ser selecionados. Cada um desses indivíduos apresenta a mesma probabilidade de seleção. Neste caso, é observado que a pressão seletiva diminui a medida em que *T* aumenta (ZEBULUM, 1999).

No método de **seleção por normalização linear**, ordenam-se os indivíduos inicialmente de acordo com sua aptidão. A seguir, estes valores são alterados de acordo com a posição relativa de cada indivíduo. Ao melhor indivíduo é atribuída uma aptidão de valor $\eta^+$ e ao pior indivíduo uma aptidão $\eta^-$. Estes dois valores são determinados pelo usuário, mas originalmente este método prevê que as condições $\eta^+ = 2\text{-}\eta^-$ e $\eta^- \geq 0$ devam ser atendidas. Os demais indivíduos têm valores de aptidão linearmente distribuídos onde $\eta^-$ *e* $\eta^+$ de acordo com sua oposição relativa na ordenação (ZEBULUM, 1999).

Finalmente, o método de seleção por **normalização exponencial** as probabilidades de seleção da cada indivíduo seguem uma função exponencial. Esta probabilidade é dada pela fórmula:

$$p_i = \frac{c-1}{c^N - 1} c^{N-i}, \qquad i \in \{1,..., N\} \tag{2.3}$$

onde c determina o grau de exponencialidade da função, podendo variar de 0 até 1.

Este trabalho baseia-se em uma das mais comuns que é a seleção por torneio e que apresenta pequeno esforço computacional.

## 2.5 OPERADOR DE RECOMBINAÇÃO (CROSSOVER)

Este operador inspira-se na idéia de recombinação de material genético.Três tipos de crossover são usualmente utilizados em AG's, o crossover de um ponto, de 2 pontos e o crossover uniforme. No primeiro, os cromossomos são cortados em um determinado ponto escolhido de forma aleatória, e os segmentos são trocados. Para o crossover de dois pontos, dois pontos são aleatoriamene escolhidos. No caso de crossover uniforme, a contribuição de cada genitor na formação dos descendentes é determinada por um padrão aleatório , o que equivale à criação de diversos pontos de corte. Os dois primeiros tipos de crossover preservam características codificadas nos cromossomos. O crossover uniforme possui a capacidade de combinar quaisquer padrões de bits, mas apresenta mais chances de destruir boas características dos cromossomos (ZEBULUM, 1999). Geralmente o crossover é aplicado com uma alta taxa - *tr* , acima de 60%.

Neste trabalho foi implementada a recombinação de um simples ponto a cada dois cromossomos selecionados, que consiste em determinar um ponto de cruzamento de maneira aleatória entre 1 e ($k$ – 1) para uma configuração de $k$ elementos, e trocar as parcelas à direita do ponto de cruzamento entre as duas configurações, conforme mostra a Figura 2.2. Cumpre lembrar que a recombinação só é efetivada se um número gerado aleatoriamente for menor que a taxa de recombinação, *tr* definida pelo usuário. Caso contrário, os cromossomos selecionados serão apenas repassados para a próxima geração ou iteração. A taxa de recombinação especifica de forma aleatória, a porcentagem de configurações que devem ser cruzadas.

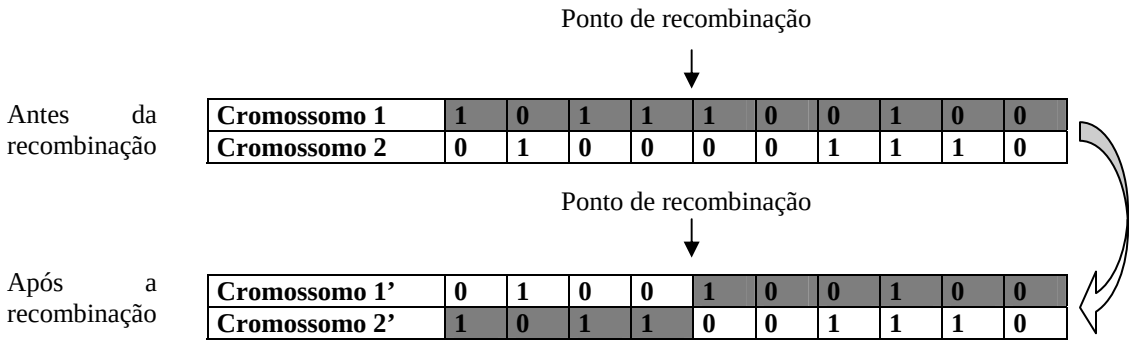



Figura 2.2 – Recombinação entre os cromossomos "pai" e "mãe", ponto de recombinação = 4.

## 2.6 OPERADOR DE MUTAÇÃO

Depois da seleção e crossover , todos os novos indivíduos da população sofrem a aplicação deste operador. O operador de mutação tem caráter exploratório, dispersando a população através do espaço de busca. A taxa de execução deste operador determina a probabilidade com que é aplicado em uma posição particular de um cromossomo. Ao ser aplicado, um novo valor é escolhido de forma aleatória, dentro dos limites do alfabeto da representação utilizada, para o alelo em questão. Na literatura de AG's (GOLDBERG, 1989), este operador é aplicado com taxas baixas, em torno de 0,1% por posição do cromossomo. Outros trabalhos vem demonstrando que em determinadas aplicações, o aumento desta taxa para valores entre 1 e 5%, melhora o desempenho dos AG's .

O operador de mutação é importante porque mantém a diversidade da população e a variabilidade dinâmica dos programas em evolução. A mutação aplicada nos exemplos é do tipo indutiva, recomendada quando se trata de representação numérica dos elementos (genes) que formam o cromossomo. Neste tipo de mutação, toma-se um cromossomo e aleatoriamente é alterado somente um elemento através de acréscimo ou redução de uma unidade. A quantidade de genes a serem alterados é definida conforme a taxa de mutação $t_m$, que é determinada de forma estocástica. Para o caso da codificação binária o operador mutação simplesmente troca o valor da variável. Se a variável escolhida for "0", troca-se para "1" e vice-versa, conforme Figura 2.3.

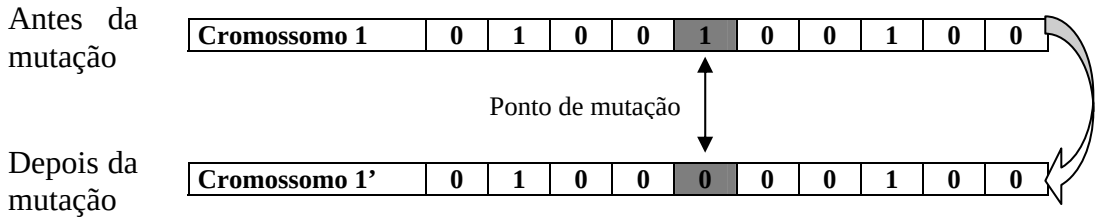



Figura 2.3 – Mutação no cromossomo.

Finalmente, depois da aplicação destes operadores substitui-se a população antiga pela nova população. Esta substituição pode ser total ou parcial. No último caso, apenas os piores indivíduos são substituídos, recebendo o nome de *Steady - State Reproduction*. Mas, outra estratégia vem sendo empregada com sucesso em diversas aplicações e foi adotada neste

trabalho que é o **elitismo**, ou seja, o melhor cromossomo de cada geração é copiado para a próxima.

## 2.7 CRITÉRIO DE PARADA

O critério de parada controla o término dos algoritmos evolutivos. Portanto, este critério dependerá basicamente do problema a ser tratado. Tendo por base alguns resultados experimentais (ZEBULUM, 1999), pode-se citar alguns critérios empregados com êxito tais como, um número máximo de iterações, indivíduos com função objetivo satisfatória, uma combinação desses critérios ou qualquer outro que satisfaça as especificações do problema.

# 3 *HARDWARE* EVOLUTIVO E A SÍNTESE DE CIRCUITOS

O desenvolvimento de técnicas novas como os algoritmos evolutivos que tem por base a evolução biológica natural, trouxeram aos pesquisadores uma ampla área de pesquisa e fonte de soluções para problemas de diferentes complexidades. Dentre essas novas técnicas estão os circuitos evolutivos que utilizam dispositivos reconfiguráveis e algoritmos evolutivos para executar tarefas em diversos tipos de aplicações, como reconhecimento de padrões (HIGUCHI et al., 1999; IWATA et al., 1996), adaptação on-line (KEYMEULEN et al., 1997; HIGUCHI et al., 1999), controle (THOMPSON, 1995), e apresenta-se como uma alternativa para síntese e otimização de circuitos eletrônicos complexos também encontrados na literatura (MANDERICK e HIGUCHI, 1997; HEMMI et al., 1995; YAO e HIGUCHI, 1997).

Neste capítulo, faz-se uma revisão bibliográfica sobre o circuito evolutivo até os dias atuais e suas perspectivas de desenvolvimento. Em seguida, apresenta-se uma breve explanação sobre os circuitos evolutivos com suas características principais, tipos de dispositivos, representação e a aplicação na síntese de circuitos digitais, tema central desta dissertação.

## 3.1 REVISÃO BIBLIOGRÁFICA

Os primeiros trabalhos utilizando algoritmos genéticos na síntese de circuitos digitais foram propostos por Louis e Rawlins em 1991 (LOUIS e RAWLINS, 1991). Nestes, os AGs são utilizados na amostragem de um espaço de busca contendo um circuito de forma e tamanho arbitrários. O AG emprega operadores de recombinação especializados para direcionar o espaço de busca. Foram implementados como caso de estudos, um detector de paridade de 5 bits e um somador de dois bits.

(KOZA, 1992; KOZA et al., 1997) utilizam programação genética em síntese de circuitos digitais combinacionais, em exemplos como multiplexadores e detectores de paridade.

Vieram outros trabalhos nesta área de pesquisa (HIGUCHI et al., 1994; KITANO, 1996), que abordavam a evolução de circuitos combinacionais frequentemente utilizados em sistemas digitais, como os exemplificados anteriormente.

(IBA et al., 1997), desenvolveram uma aplicação utilizando algoritmos evolutivos na síntese de detectores de paridade e multiplexadores, mapeando os circuitos em FPLAs (*Field Programmable Logic Arrays)*. Os FPLAs têm a sua estrutura limitada a portas lógicas AND e OR, porém neste procedimento os circuitos sintetizados podem ser diretamente implementados em dispositivos existentes comercialmente.

(HIGUCHI et al., 1999), aplicaram evolução de circuitos digitais na área de reconhecimento de padrões. O trabalho de (KAJITANI et al., 1998) aplica circuitos evolutivos no controle de uma prótese de mão. Usa o AG para sintetizar um circuito digital diretamente em uma arquitetura PLA (*Programmable Logic Array*) (SANCHES, 1996) através de uma seqüência de padrões de entrada, sinais elétricos musculares, e o circuito sintetizado, chip evolutivo, gera saídas utilizadas no controle da mão artificial.

(IWATA et al., 2001), implementam um dispositivo com a técnica evolutiva onde a arquitetura em hardware de um AG, uma lógica reconfigurável e uma lógica de controle são programadas dentro do mesmo dispositivo. Este experimento foi aplicado para controlar os seis movimentos de uma prótese de mão, por exemplo, abrir e fechar a mão, virar a mão para a direita e esquerda, e os movimentos de flexão e extensão. Este trabalho viabiliza aplicações que necessitam de intervenções on-line na reconfiguração de hardware.

(SUSHIL, 1993) utiliza o algoritmo genético para realizar a ligação de portas lógicas e sintetizar um circuito combinacional otimizado, usando a representação em níveis, e a substituição de portas lógicas por fios, sem a perda de sua funcionalidade original.

O Algoritmo Genético também é utilizado em (COELLO et al., 2001) com um tamanho variável de cromossomos, obtendo os circuitos na condição de portas lógicas. Onde compara seus resultados com a otimização através do Mapa de Karnaugh e o Método de Quine-McCluskey, além de configurar os cromossomos tanto em binários como números inteiros.

Em (MANTOVANI et al., 2004) relata a síntese de circuitos usando a técnica de hardware evolutivo e reconfiguração em dispositivos PLAs usando representação por mapa de fusíveis e o algoritmo genético.

(TYRREL et al., 2004) mostra um algoritmo evolutivo aplicado no controle direcional de um robô, adaptando seu hardware para evitar colisões ou falhas acusadas por seus sensores.

Em (CAMPOS et al., 2004) foi desenvolvido um EHW que utiliza a programação genética e cromossomos formados por expressões simbólicas LISP com comprimentos variáveis, para a obtenção de projetos de circuitos lógicos combinacionais simples, como detectores de números primos de 4 e 6 bits.

(JEWAJINDA; CHONGSTITVATANA, 2006) mostram um algoritmo genético modificado denominado COCGA (*Cooperative Compact Genetic Algorithms*) realizando um método de processamento paralelo para vários AG compactos. Esses AG compactos evoluem com dois indivíduos e realizam os operadores genéticos de forma a manter uma constante evolução. Com isso mostram melhorias tanto na qualidade do resultado como na velocidade de convergência.

(STOMEO et al., 2006) propõem um algoritmo desenvolvido para a aplicação em PLA's com mecanismos de avaliação, seleção e recombinações próprios para a síntese de circuitos. O processo se inicia com a criação do cromossomo baseado na estrutura de dispositivos reprogramáveis. O processo de validação é fundamentado na tabela verdade do circuito a ser desenvolvido e a seleção utiliza dois estágios, no primeiro estágio são selecionados os melhores indivíduos para participarem da recombinação e no segundo estágio é realizada uma mistura entre o restante da população para completar a próxima população na evolução. Com uma codificação binária e aplicação extrínseca revela-se um algoritmo genético que realiza a síntese de circuitos lógicos com poucas iterações (138 em média) e reduz o circuito proposto em 40%.

(KITANO, 1996) codifica os circuitos no modo matricial denominado por ele de *grammar encoding method*, e mostra que os resultados são diretamente afetados pelas alterações no fenótipo e genótipo.

(KAJITANI et al., 1996), propõem a utilização de cromossomos de tamanho variado para a evolução do hardware, o que possibilitou a evolução de circuitos combinacionais e o reconhecimento de padrão.

(KEYMEULEN et al., 2000) aborda sistemas com tolerância à falhas usando a técnica de circuito evolutivo com a codificação dos cromossomos em binários para serem aplicados ao FPTA (*Field Programmable Transistor Array).* Aplica seleção, recombinação de um simples ponto e a mutação apenas complementa o alelo selecionado.

(HEREFORD, 2004) também utiliza a técnica EHW para a detecção de falhas, mas em sensores de temperatura. O algoritmo genético básico implementado usa recombinação, mutação, e seleção proporcional (*rollete wheel*).

Em (LEE et al., 2000) é apresentado um trabalho com programação genética aplicada de forma *on-line* (no próprio hardware) para o movimento de robôs, salientando as dificuldades de realizar o operador de recombinação e a codificação por árvore genética.

Aplicado à nanotecnologia (VILELA NETO et al., 2006) relata o EHW em três diferentes aplicações: síntese de transistores moleculares simulados no *software* SPICE, síntese e otimização de QCA ( *Quantun-dot Cellular Automata*) e a otimização de Diodos Orgânicos Emissores de Luz - OLED ( *Organic Light-Emitting Diodes)*. Mostra que a técnica do EHW é viável para tratar de dispositivos orgânicos como o OLED.

Em resumo, as aplicações do EHW se concentram em reconhecimento ou classificação de padrões, tolerância à falhas, síntese de projetos digitais ou analógicos, controle e robótica sendo também utilizado em VLSI (*Very Large Scale Integration*) (ZEBULUM et al., 1997).

A abrangência dos temas citados forneceu os subsídios necessários para o desenvolvimento desta pesquisa, e proporcionou conhecimentos através de experimentos, para que novas aplicações sejam realizadas.

## 3.2 *HARDWARE* EVOLUTIVO

Hardware Evolutivo é uma teoria baseada em algoritmos evolutivos e dispositivos lógicos programáveis e vem sendo usada no desenvolvimento de projeto de circuitos analógicos ou digitais onde são crescentes a síntese e otimização de circuitos complexos.

Denomina-se Hardware Evolutivo ao circuito que modifica sua própria estrutura e comportamento por interação com o meio. Utilizam algoritmos evolutivos (EA) como seu mecanismo principal de adaptação e os dispositivos reconfiguráveis, como plataforma de evolução (HIGUCHI et al., 1993; YAO e HIGUCHI, 1997).

Dispositivos reconfiguráveis típicos são os que consistem de um conjunto de portas AND e OR, como o PLA (*Programmable logic Array*) e o FPGA (*Field programmable Gate Array*).

São considerados algoritmos evolutivos, o Algoritmo Genético (AG), Programação Genética (PG), Programação Evolutiva (PE) e Estratégia Evolutiva (EE). Mais

detalhes destes algoritmos são encontrados em (ZEBULUM, 1999). Na tabela 3.1 são resumidas as características principais de cada um destes algoritmos. Todavia, um dos mais utilizados para evoluir circuitos é o Algoritmo Genético (ZEBULUM et al., 1997). Neste trabalho foi utilizado o algoritmo genético como técnica evolutiva.

Tabela 3.1 – Características dos algoritmos evolutivos (VON ZUBEN e CASTRO, 2004).

| | **AG** | **EE** | **PE** | **PG** |
|---|---|---|---|---|
| **Representação** | Cadeias binárias | Vetores reais | Vetores reais | Árvores |
| **Auto-adaptação** | Nenhuma | Desvio padrão e co-variâncias | Desvio padrão e coeficiente de correlação | Nenhuma |
| **fitness** | Valor escalonado da função objetivo | Valor da função objetivo | Valor (escalonado) da função objetivo | Valor escalonado da função objetivo |
| **Seleção** | Probabilística | Determinística | Determinística | Probabilística |
| **Recombinação** | Principal operador | Diferentes variações, importante para a auto-adaptação | Nenhuma | Um dos operadores |
| **Mutação** | Operador secundário | Principal operador | Único operador | Um dos operadores |

O circuito evolutivo considera a arquitetura de bits de um dispositivo reconfigurável como um cromossomo para um AG, que busca por uma estrutura otimizada de circuito.

O cromossomo do AG, ou arquitetura de bits, programa um dispositivo reconfigurável durante a aprendizagem genética, devido a este fato os EHW podem ser considerados como um circuito adaptativo on-line.

São várias as definições e classificações propostas por inúmeros pesquisadores, para o processo do hardware evolutivo, mas todas elas mostram um consenso que é apresentado em (ZEBULUM et al. 1997), e que podem ser resumidas em três tópicos:

1. Processo de Avaliação do Hardware;
2. Algoritmo Evolutivo;
3. Dispositivos para a Evolução;

Processo de Avaliação do Hardware

A avaliação do hardware pode ser feita de forma extrínseca ou intrínseca. No modo intrínseco a evolução acontece no dispositivo reprogramável, sendo repassados e avaliados os cromossomos no hardware. Em outras palavras, a avaliação é feita com base no

comportamento dos indivíduos ao serem carregados nos dispositivos reconfiguráveis. Esse método também é chamado de on-line, e apresenta a vantagem de ser mais rápido e não necessitar de nenhum modelo de simulação.

Para o caso do hardware extrínseco, a avaliação é feita via software, em outro ambiente e ao término do processo de evolução é repassado somente o melhor cromossomo ao PLD, ou seja, os circuitos são avaliados através de simuladores.

Algoritmo Evolutivo

O algoritmo evolutivo caracteriza a estratégia evolutiva usada para evolução do EHW. As quatro formas artificiais de evolução do hardware mais comuns são o algoritmo genético, a programação genética, programação evolutiva e a estratégia evolutiva, já apresentadas na tabela 3.1. Mas, existem outras diferentes formas de algoritmos evolutivos que são VGA's ( *Variable Length Chromosomes Genetic Algorithms)* e PGA's ( *Production Genetic Algorithms)*, porém pouco utilizadas (ZEBULUM et al., 1997).

Dispositivos para a Evolução

Dispositivos para evolução ou plataforma do EHW conforme a definição são utilizados os circuitos lógicos programáveis como PLD's simples, CPLD (*Complex Programmable Logic Devices*) ou FPGA's (*Field Programmable Logic Arrays*). Mais recentemente, outros dispositivos programáveis estão sendo usados, por exemplo, o FPTA (*Field-Programmable Transistor Arrays*) (KEYMEULEN et al., 1997), FPAA (*Field-Programmable Analog Array*) ou circuitos integrados dedicados ASICs( *Applications Specific Integrated Circuits*) ou seja, projetado para um determinado usuário.

## 3.3 SÍNTESE E OTIMIZAÇÃO DE CIRCUITOS

Para atender o aumento da complexidade de projetos dos circuitos tanto analógicos quanto digitais, é crescente o aumento das pesquisas na área de síntese e otimização de cicuitos. Cada vez mais, busca-se por ferramentas eficientes, não somente para implementação de funções e a minimização do circuito, mas também para otimizar fatores como tempo de execução e comercialização do projeto. Os circuitos evolutivos, que aliam dispositivos reconfiguráveis e técnicas modernas de otimização fornecem uma alternativa

interessante, baseadas na computação evolutiva, competitivas com o estado da arte em eletrônica.

Os sistemas evolutivos têm sido aplicados à otimização de circuitos integrados, a partir de um projeto inicial do circuito funcional, mas não otimizado, fornecido como entrada em problemas como otimização de funções lógicas; mapeamento em dispositivos lógicos programáveis; roteamento global; floorplanning (ZEBULUM, 1999). Estas aplicações tem demonstrado eficiência em termos de qualidade de solução, embora raramente sejam competitivas em termos de tempo de execução.

A disponibilidade de uma grande quantidade de simuladores para realizar experimentos extrínsecos e a disponibilidadede dos dispositivos programáveis (FPGAs), motivam a síntese de circuitos digtais e a otimização de funções lógicas.

A síntese de um circuito inicia-se pela descrição de um problema, ou de sua especificação funcional, buscando-se uma topologia adequada para aplicação prática, escolha de tipos e valores de componentes, e se for o caso, a otimização do circuito (topologia mínima).

A síntese de circuitos mostra-se como uma tarefa mais complexa do que a de otimização, uma vez que envolve aspectos físicos, técnicos e econômicos, tornando complexa a modelagem do problema.

Os métodos tradicionais para projetar circuitos lógicos combinacionais minimizados, dentre eles, Mapas de Karnaugh - MK, Karnaugh (1953), Quine-McCluskey, McCluskey (1956), programa ESPRESSO que utiliza Quine McCluskey, etc. realizam a síntese a partir de uma tabela verdade, fornecendo circuitos otimizados em número de portas e variáveis de entrada para as portas.

Os dois primeiros métodos serviram como base de comparação para os testes realizados nesta pesquisa.

Nesta dissertação, avalia-se o desempenho do hardware evolutivo extrínseco aplicado à síntese de circuitos digitais combinacionais básicos, que realize uma determinada função, a partir de uma tabela verdade ou expressão lógica equivalente, usando um algoritmo genético.

### 3.3.1 Métodos de Minimização Clássicos

#### Mapas de Karnaugh

É um processo de minimização de funções booleanas que faz uso de uma representação gráfica, baseando-se na tabela verdade. Torna-se uma tarefa árdua e com grandes chances de erros ao minimizar funções com 6 ou mais variáveis, pois seu resultado depende da experiência de quem executa a busca das paridades a serem cobertas (implicantes). Para ilustrar este processo apresenta-se na Figura 3.2 um exemplo de minimização pelo mapa de Karnaugh. A Figura 3.1 mostra a representação em portas lógicas da função original (1) e da função minimizada (2).

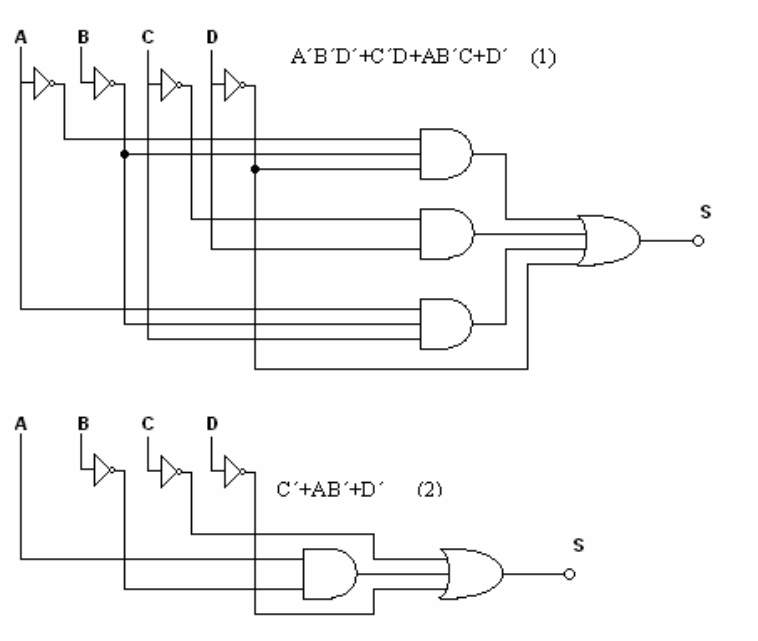



Figura 3.1 - Representação em portas lógicas

A figura 3.2 mostra como foi feita a minimização dessa função no MK.

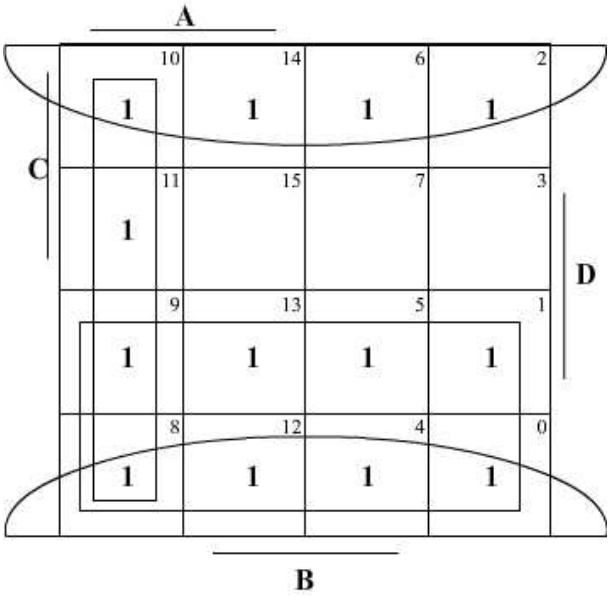



Figura 3.2 – Mapa de Karnaugh

### 3.3.2 Método de Quine-McCluskey

Esse processo minimiza as funções em forma de soma de produtos através de um método tabular. Sua busca baseia-se na geração de todos os implicantes, o que aumenta exponencialmente a sua complexidade conforme aumentam as variáveis de entrada.

A minimização pelo método de Quine-McCluskey baseia-se no seguinte algoritmo:

1. Produzir a expansão dos mintermos (na forma padrão da soma de produtos) para a função a ser minimizada;
2. Eliminar tantas literais quanto possíveis pela sistemática de $XZ + XZ' = X$;
3. Utilizar um gráfico de implicantes primos para selecionar um conjunto mínimo de implicantes primos que quando em forma de soma de produtos juntos, produzem uma minimização que contêm o número mínimo de literais.

### 3.3.3 Programa ESPRESSO

É um processo computacional de minimização de funções booleanas, fundamentado no método de Quine-McCluskey, que realiza a cobertura do problema sem a necessidade de gerar todo o conjunto de implicantes primos, reduzindo assim o tempo computacional. O programa lê um arquivo de entrada com a representação lógica do conjunto

de equações booleanas compatível com o padrão Berkeley como ilustrado na Figura 3.3 e a execução do programa é por linha de comandos em ambiente DOS. Tais comandos, definem os métodos a serem aplicados na minimização dentre as opções disponíveis do programa (MCGEER et al., 1993).

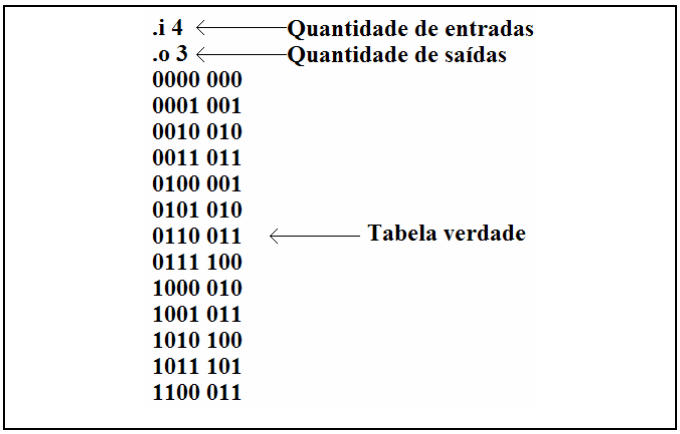



Figura 3.3 – Arquivo de entrada no padrão Berkeley.

## 3.4 REPRESENTAÇÃO

A representação é um fator crítico quando se utiliza a técnica evolutiva e está relacionada à forma com que os circuitos são codificados em um cromossomo. Uma representação incorreta pode levar a soluções inválidas ou infactíveis e não atender às especificações iniciais do projeto.

Pode-se usar uma representação direta onde a cadeia de bits especifica a conectividade e funções de diferentes componentes de hardware do circuito (frequentemente a níveis de portas) ou representação indireta onde o cromossomo é representado como árvores ou símbolos gramaticais, usados para gerar circuitos, caso de Programação Genética (CAMPOS et al., 2004).

Portanto, neste trabalho são apresentadas três formas de representação diretas, uma que utiliza soma de produtos das variáveis lógicas da função combinacional, onde os cromossomos são codificados em uma coleção de genes para representar todos os possíveis produtos de variáveis lógicas completas ou não.

A Figura 3.4 ilustra essa codificação, que mostra três números possíveis para o gene. O ‘0’ que é o complemento da variável, o ‘1’ que é a presença da variável e o ‘2’ que representa a ausência de uma determinada variável, definindo assim o alfabeto ou espaço

cromossomial em $3^9$, onde 3 é o número de genes e 9 é o número total contido no cromossomo.

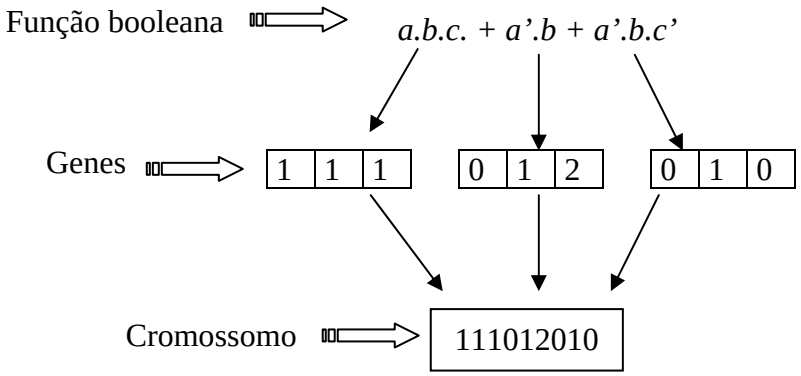



Figura 3.4 – Representação através de funções booleanas (soma de produtos).

Uma outra forma de representação que pode ser usada é a codificação do circuito por portas lógicas onde um cromossomo representa as entradas e as funções lógicas por números inteiros, necessária à subdivisão do cromossomo em níveis que determinam o circuito.

A Figura 3.5 ilustra esta representação através de um exemplo de circuito lógico em três níveis, sendo definidas cinco portas lógicas de duas entradas (AND, NAND, OR, NOR e XOR) codificando-as de 0 a 4, determinando oito entradas possíveis de E1 até E7 e uma saída. As entradas podem ser sinais vindos de um sinal externo. A quantidade de níveis e o número de portas para cada nível são determinados pelo projetista.

Com essa representação por números inteiros e portas lógicas, diminui a quantidade de genes do cromossomo e com isso aumenta a eficiência do algoritmo.

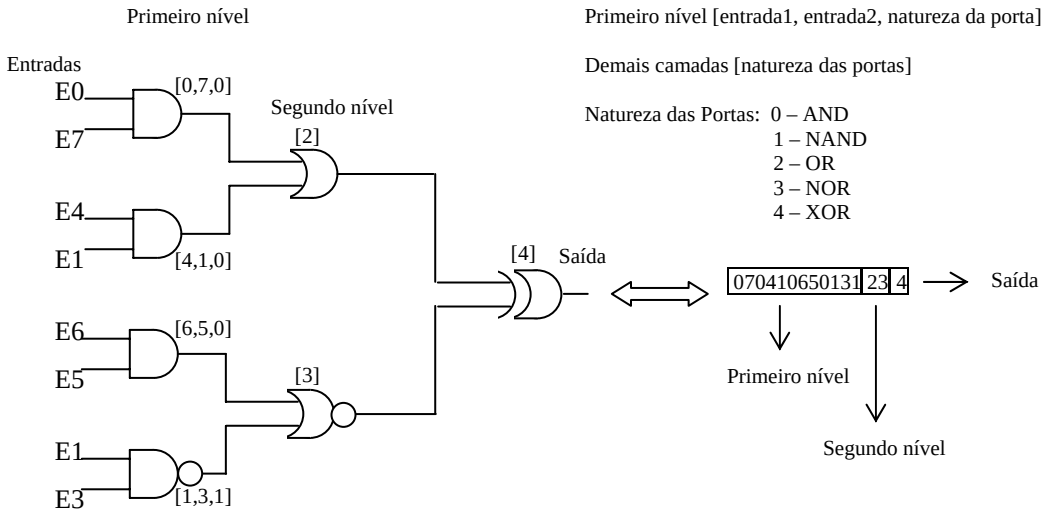



Figura 3.5 – Mapeamento entre circuitos e cromossomos (ZEBULUM et al., 1997).

O terceiro exemplo de codificação mostrado é a forma de mapeamento de fusíveis encontrada em (HIGUSHI et al., 1993) que se baseia na arquitetura de um dispositivo lógico programável - *Programmable Logic Array* (PLA), formado internamente de uma matriz de funções AND-OR programáveis e conexões de fusíveis, sendo capaz de representar qualquer função combinacional, como mostrado em Mantovani (MANTOVANI e OLIVEIRA, 2004).

A Figura 3.6 mostra a idéia desta representação para um circuito lógico combinacional, o decodificador com duas entradas (A0, A1) e quatro saídas (Y0, Y1, Y2,Y3). As conexões ou fusíveis ($f_i$) compõem as linhas de fusíveis ($p_0$, $p_1$, $p_2$, $p_3$). Segue uma representação binária que define zero (0) como um fusível inativo e um (1) para ativo ou conectado.

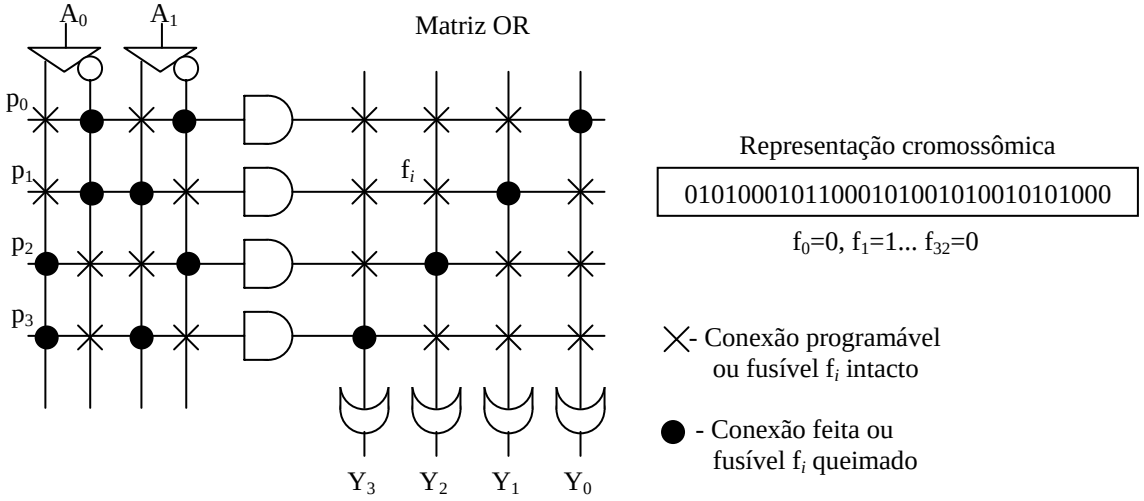



Figura 3.6 – Mapeamento de fusíveis e sua representação cromossômica

Por simulação do mapa de fusíveis ou pontos de conexão, é efetuada a avaliação (*fitness*) do AG como se fosse o próprio circuito. Esta representação restringe-se às aplicações em circuitos combinacionais que são descritos por somas de produtos ou conjuntos completos e a avaliação é feita somente por sua tabela verdade ou aspecto funcional do mesmo.

Para o dimensionamento do mapa de fusíveis são necessárias as especificações do circuito, ou seja, do número de entradas ***n***, do número de termos produto ***k*** e o número de saídas ***m.*** A quantidade de conexões programáveis no plano de entrada é igual ***2.n.k*** e no plano de saída é igual a ***k.m*** ou um total de número de genes no cromossomo igual a

$$Ng = 2 \cdot n \cdot k + k \cdot m \qquad (3.1)$$

Em outras palavras ***n*** determina a quantidade de *buffers* inversores, ***k*** a quantidade de portas AND e ***m*** determina o número de portas OR.

Encontram-se na literatura de síntese de circuitos, outros modelos de representação, como por exemplo usando transistores, (ZEBULUM, 1999) e expressões simbólicas Lisp, para uso em programação genética, (CAMPOS et al., 2004). A escolha do modelo pode ser baseada na complexidade do circuito que se deseja projetar.

# 4 REPRESENTAÇÃO E TÉCNICAS DE SOLUÇÃO ABORDADAS

Neste capítulo serão mostradas as técnicas de solução envolvendo a representação e os tipos de operadores para implementar o Algoritmo Genético, aplicadas na síntese de circuitos combinacionais e seqüenciais abordadas nesta dissertação.

Inicialmente, foram realizados estudos para a síntese de circuitos a partir de uma representação por mapas de fusíveis conforme mostrado no capítulo anterior. Este tipo de representação por ser aplicada a dispositivos lógicos programáveis, não realiza uma otimização do circuito.

Buscando não somente a síntese, mas também a otimização de circuitos mais complexos optou-se por usar portas lógicas, e uma codificação binária, conforme proposto por (COELLO et al. 2001; SUSHIL, 1993).

Com o objetivo de melhorar o desempenho do AG, usou-se também uma representação por portas lógicas e codificação por números inteiros. No texto que segue, são mostrados os algoritmos para as técnicas mencionadas.

## 4.1 IMPLEMENTAÇÃO POR MAPAS DE FUSÍVEIS

A implementação por mapas de fusíveis simula a arquitetura interna de um PLA, que possui no seu interior uma matriz de funções AND-OR programáveis, sendo capaz de representar qualquer função combinacional. Outros dispositivos lógicos como FPGAs podem implementar funções seqüenciais que exigem componentes mais sofisticados como *flip-flops* e memórias.

O processo de síntese inicia-se com a descrição do circuito ou sua tabela verdade. Com base nessas informações o AG terá condições através da sua função objetivo, de avaliar, a fim de convergir para o circuito correto. Para ilustrar este tipo de representação foi realizado um exemplo para o somador inteiro.

*Somador Inteiro*

O Somador inteiro é um circuito combinacional que realiza a soma completa de uma coluna de números binários $A_i$ e $B_i$ e o vai um de entrada $C_i$. Sua tabela verdade é expressa através da Tabela 4.1 que apresenta como saída a soma $S_i$ e o vai um de saída $C_{i+1}$.

Tabela 4.1 - Tabela verdade para o somador inteiro.

| **A** | **B** | **$C_i$** | **S** | **$C_{i+1}$** |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Conforme teoria de dimensionamento apresentada no capítulo 3, o mapa de fusíveis e portanto o cromossomo deve ter 56 genes (3 entradas, 3 inversores, 7 portas AND’s e 2 saídas). O mapa de fusíveis para o somador binário inteiro é mostrado na Figura 4.1. Definida a representação do circuito em cromossomo, toma-se como função objetivo para este exemplo o total de acertos especificados pelas linhas (8) da tabela verdade.

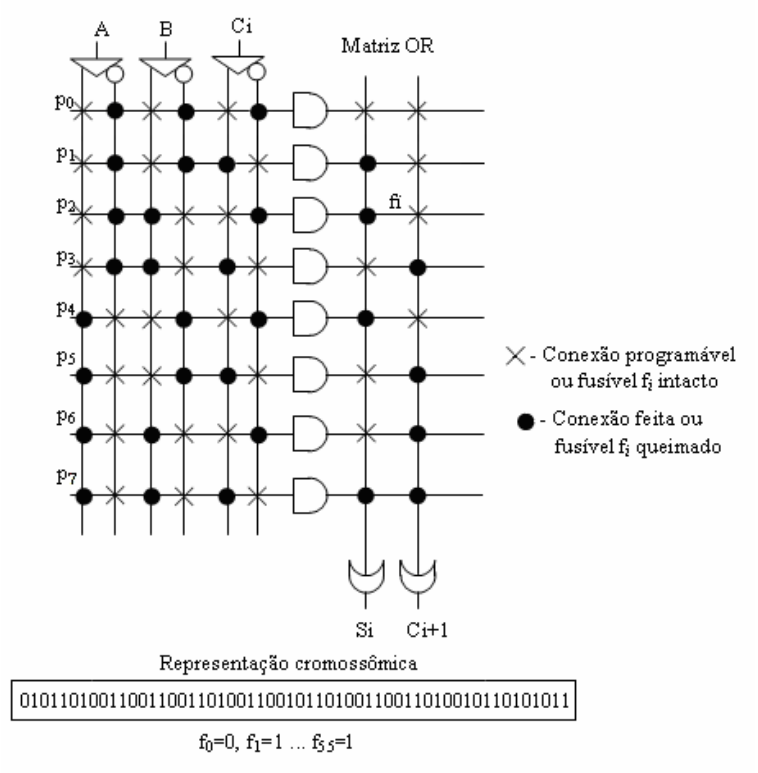



Figura 4.1 – Mapa de fusíveis para o somador inteiro.

Nota-se na representação cromossômica apenas 56 genes, pois o programa criado ignora as entradas do circuito que possuem a saída “00”, que no caso do somador é a entrada “000”. O objetivo de ignorar tais entradas é reduzir o tempo de processamento, pois a ausência dessa entrada não impede a representação correta do circuito.

As linhas da tabela verdade no mapa, são designadas por ‘$P_i$’ e as saídas são designadas por ‘$Y_i$’. No caso do somador inteiro, são necessários $P_0$ a $P_7$, e as duas saídas $Y_0$ e $Y_1$.

As variáveis $P_i$ definem o acerto do cromossomo com a parte que representa as entradas, e as variáveis $Y_i$ confrontam as saídas com os $P_i$ encontrados. Esses dados são conferidos com a tabela verdade para assinalar a pontuação do cromossomo.

A equação (4.1) representa a avaliação do cromossomo com a tabela verdade na parte da entrada, onde os símbolos extraídos da linguagem de programação C por convenção são:

- **exclamação** (!) que representa uma negação;
- **ponto** (.) representa uma porta AND ;
- **barra** ( | ) representa a porta OR.

A tabela verdade é constituída por linhas e colunas, portanto para confrontar os valores do cromossomo é preciso definir qual parte da tabela verdade está sendo cruzada, de forma que TV [ i ] [ j ], é a variável onde *i* é a linha e *j* é a coluna.

$$P_i = !(((fi \ \& \ !(TV[i][j])) \mid (fi+1 \ \& \ TV\ [i]\ [j])) \mid ((fi+2 \ \& \ !(TV[i][j+1])) \mid (fi+3 \ \& \ TV\ [i]\ [j+1]))) \quad (4.1)$$

A entrada testada por $P_i$, agora é verificada com os genes do cromossomo que representam a saída. Através da equação 4.2 avaliam-se as saídas do cromossomo com os $P_i$ testados, resultando nas saídas do circuito. Essas saídas são comparadas às saídas da tabela verdade e caso sejam iguais, é marcado o ponto para o cromossomo.

$$Yi = ((fi+4 \ \& \ Pi) \mid (fi+12 \ \& \ Pi+1) \mid (fi+20 \ \& \ Pi+2) \mid (fi+28 \ \& \ Pi+3)) \quad (4.2)$$

Para ilustrar essa definição mostra-se no exemplo a seguir a avaliação de parte do cromossomo representado na Figura 4.1.

$P_1 = !(((f_0 \ \& \ !(TV[2][1])) \mid (f_1 \ \& \ TV\ [2]\ [1])) \mid ((f_2 \ \& \ !(TV[2][2])) \mid (f_3 \ \& \ TV\ [2]\ [2]))) \mid (f_4 \ \& \ TV\ [2]\ [3]))) \mid (f_5 \ \& \ TV\ [2]\ [3])))$

$P_1 = !\ (((0 \ \& \ !(0)) \mid (1 \ \& \ 0)) \mid ((0\& \ !(0)) \mid (1 \ \& \ 0)) \mid ((1\& \ !(1)) \mid (0 \ \& \ 1)))$

$P_1 = !\ ((0 \mid 0) \mid (0 \mid 0) \mid (0 \mid 0))$

$P_1 = !\ (0 \mid 0 \mid 0)$

$P_1 = 1$

Como para essa parte do cromossomo a saída ativa é a $Y_0$ realizar-se-á o teste com esta saída, porém se tem em mente que as outras saídas são testadas, pois poderia estar representada qualquer uma das linhas da tabela verdade.

$Y_3 = ((f_6 \ \& \ P_0) \mid (f_{14} \ \& \ P_1) \mid (f_{22} \ \& \ P_2) \mid (f_{30} \ \& \ P_3) \mid (f_{38} \ \& \ P_4) \mid (f_{46} \ \& \ P_5) \mid$

$(f_{54} \ \& \ P_6) \mid (f_{62} \ \& \ P_7))$

$Y_3 = ((0\& \ 1) \mid (1\& \ 1) \mid (1 \ \& \ 1) \mid (0\& \ 1) \mid (1 \ \& \ 1) \mid (0 \ \& \ 1) \mid (0 \ \& \ 1) \mid (1 \ \& \ 1))$

$Y_3 = (0 \mid 1 \mid 1 \mid 0 \mid 1 \mid 0 \mid 0 \mid 1)$

$Y_3 = 1$

Foi obtida a saída desejada, portanto o ponto foi marcado validando essa parte do cromossomo avaliado. Esse processo de avaliação é realizado para todas as partes do cromossomo ($P_0$ a $P_7$).

Pode-se encontrar uma representação da terceira linha da tabela verdade no começo do cromossomo, sendo necessária a verificação de todas as partes do cromossomo com todas as linhas da tabela verdade. Caso todas as partes do cromossomo não fossem testadas com todas as linhas da tabela, resultaria em um desperdiço de esforço computacional, porque um cromossomo bom seria mal avaliado.

## 4.2 IMPLEMENTAÇÃO POR PORTAS LÓGICAS

### 4.2.1 Circuito Combinacional

Codificação

Inicialmente, mostra-se a modelagem do circuito evolutivo realizado de modo extrínseco para um circuito combinacional.

Originalmente, cada cromossomo tem o formato de um vetor ou *string* binário que representa as entradas do circuito e as funções realizadas por cada porta. Esta codificação binária foi transformada para uma codificação de números inteiros, tendo em vista a redução do espaço de busca e o tempo de convergência do algoritmo. Baseado em (COELLO et al., 2001) e (SUSHIL, 1993) foi usada também a representação de portas lógicas na forma matricial linear numérica, e a subdivisão do cromossomo em níveis que determinam o circuito, conforme se observa na Figura 4.2, que permite também, a variação do comprimento do cromossomo de modo a se obter um circuito correto e minimizado. A quantidade de níveis e o número de portas para cada nível são determinados pelo projetista, baseando-se em dados do circuito como o número de entradas e saídas.

A Tabela 4.1 mostra a codificação usada e a equivalência das portas. Na coluna 1, da tabela, mostra-se o número decimal associado. Na coluna 2, os três primeiros bits da esquerda para a direita definem a porta, e o quarto bit determina como serão realizadas as conexões da porta (restringe-se o modelo a portas de duas entradas). Na coluna 3 estão associados os tipos de portas que estão sendo utilizadas.

Nesta representação os estudos foram restritos a uma matriz bidimensional e cada elemento da matriz é uma porta lógica limitada a duas entradas (NOT, AND, OR e XOR) ou fio de ligação (FIO). Portanto, três bits são necessários para representar até oito tipos de

portas (GOULART SOBRINHO e MANTOVANI, 2006a), (GOULART SOBRINHO e MANTOVANI, 2006b) e (MANTOVANI e GOULART SOBRINHO, 2006).

Tabela 4.2 - Codificação das Portas Lógicas.

| **Decimal** | **Binário** | **Representação** |
|---|---|---|
| 0 | 000 0 e 000 1 | FIO |
| 1 | 100 0 e 100 1 | NOT |
| 2 | 001 0 | AND Segunda entrada abaixo |
| 3 | 010 0 | OR Segunda entrada abaixo |
| 4 | 011 0 | XOR Segunda entrada abaixo |
| 5 | 001 1 | AND Segunda entrada acima |
| 6 | 010 1 | OR Segunda entrada acima |
| 7 | 011 1 | XOR Segunda entrada acima |

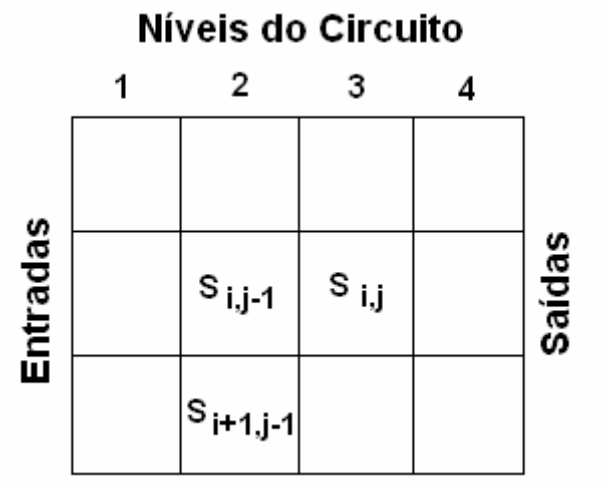



Figura 4.2 - Representação Matricial do Circuito.

O circuito é estruturado de forma que uma porta receba nas suas entradas qualquer porta da coluna anterior. Formalizando esta representação, tem-se:

Se uma porta *S* está na posição $S_{i,j}$, onde *j* indica o nível da porta (coluna) e *i* indica a posição na linha, uma entrada é proveniente da posição $S_{i,j-1}$ (mesma linha e coluna anterior); a outra entrada vem de $S_{i+1,j-1}$ (coluna anterior e uma linha abaixo) ou $S_{i-1,j-1}$,

(coluna anterior e uma linha acima). Com exceção para os extremos, ou seja, para elementos da primeira, $S_{1,j}$ e última linha, $S_{n,j}$. Nestas condições, para os elementos da primeira linha, a primeira entrada vem da posição $S_{1,j-1}$ e a segunda entrada vem da posição $S_{n,j-1}$ caso seja solicitada a entrada da coluna anterior e uma linha acima. Para elementos da última linha $S_{n,j}$, a primeira entrada vem da posição $S_{n,j-1}$, e a segunda entrada é obtida da posição $S_{1,j-1}$, caso seja necessária a entrada da coluna anterior e uma linha abaixo, para *i=1,...,n* e *j=1,..,k,* onde *n* e *k,* são números inteiros, e respectivamente valores máximos de linha e de coluna, podendo ser diferentes um do outro.

Parâmetros e Restrições

O tamanho da população e o número de interações ou gerações variam, dependendo do espaço de busca do problema a ser solucionado, e ambos são fatores que determinam o esforço computacional. O número de iterações pode servir também, como critério de parada para o algoritmo genético.

Função de Adaptação

A função de avaliação é realizada em duas etapas. A primeira avaliação visa a obtenção de um circuito lógico correto através do cálculo da soma de acertos da seqüência de saída de uma tabela verdade ou expressão booleana, de acordo com a equação matemática 4.3:

$$f(x) = \sum_{i=0}^{n-1} x_i \qquad (4.3)$$

onde, a função *f(x)* indica o número de acertos na tabela verdade do caso sobre estudo, $x_i$=0, falso; $x_i$=1, verdadeiro) e *n* é o número de linhas da tabela verdade.

Obtido o circuito que execute toda a tabela verdade ou seja, factível, busca-se na segunda etapa a minimização em termos de entradas das portas e o número de portas lógicas no circuito. Para isso, trocam-se as portas lógicas por fios enquanto o desempenho especificado pela tabela verdade do circuito seja mantido. Desta forma, aumentando o número de fios obtém-se uma função *w(x)* que é um valor inteiro igual à quantidade de fios adicionados ao circuito. Este procedimento de adicionar fios no lugar de componentes, equivale a minimizar o circuito em termos de portas lógicas encontrada em Coello et al (COELLO et al., 2001) e (SUSHIL, 1993). Portanto, a função adaptação pode ser reescrita pela equação 4.4:

$$fa(x) = f(x) + w(x) \tag{4.4}$$

Operadores Genéticos

*Seleção*

Definida a estrutura do cromossomo para o AG, uma população inicial é gerada de forma aleatória. Com base na função adaptação aplica-se o operador seleção. O operador seleção determina quais indivíduos participarão da recombinação, sendo aplicada a seleção por torneio. Juntamente com este operador, fez-se uso do elitismo, que tem como objetivo repassar as melhores configurações às próximas gerações. No elitismo implementado separa-se, com base no *fitness*, o melhor indivíduo. A seleção por torneio é realizada para o conjunto de indivíduos, através de ***n*** jogos (***n*** o tamanho da população). Neste esquema são selecionados aleatoriamente alguns indivíduos, neste caso três, e o melhor deles é escolhido. Este processo é repetido para selecionar o outro indivíduo que vai participar da recombinação.

*Recombinação*

A partir das configurações selecionadas, deve-se formar os pares de indivíduos e realizar o cruzamento, gerando dois novos descendentes. Implementou-se a "*recombinação de um simples ponto*" que consiste em determinar um ponto de cruzamento de maneira aleatória

entre *1* e (*k – 1*), para uma configuração de *k* elementos, e trocar as parcelas à direita do ponto de cruzamento entre as duas configurações, obedecendo a taxa de recombinação $t_r$ definida pelo usuário. Caso contrário, os indivíduos selecionados serão apenas repassados para a próxima geração.

*Mutação*

O operador de mutação mantém a diversidade da população. A mutação aplicada no trabalho é do tipo indutiva, recomendada quando se trata de representação numérica dos elementos (genes) que formam o cromossomo. Em um cromossomo é aleatoriamente alterado um elemento através de acréscimo ou redução de uma unidade, obedecendo a taxa de mutação $t_m$, que é determinada de forma estocástica.

A estratégia de mutação implementada avalia todos os genes no cromossomo com a taxa de mutação $t_m$ (GOLDBERG, 1989), mas para reduzir o esforço computacional deste processo, calcula-se a quantidade de genes a serem alterados pela mutação usando a equação 2.1 ($muta = tm \cdot (Tc \cdot N)$)., onde ***muta*** é a quantidade de genes que serão alterados na população, com a taxa de mutação variando de 1% a 5% , ***Tc*** indica a quantidade de genes do cromossomo i e ***N,*** o número de indivíduos da população. Definida a quantidade de genes, escolhe-se de forma aleatória, quais os genes que sofrerão a mutação.

Este processo de síntese é demonstrado através de um exemplo, o somador completo, que realiza a soma entre dois números binários (A e B) de um bit cada um, conforme tabela verdade.

4.1 Tabela Verdade para o somador

| **A** | **B** | **$C_i$** | **S** | **$C_{i+1}$** |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Que através do Mapa de Karnaugh, tem-se como resposta para a saída, $S = A \oplus B \oplus C_i$ e $C_{i+1} = BC_i + AC_i + AB$, onde $C_i$ é o vai um de entrada e $C_{i+1}$ o vai um de saída.

Para ilustrar a técnica proposta inicia-se por determinar o tamanho do cromossomo, $\boldsymbol{T_c}$ que pode ser definido pela equação (4.5).

$$T_c = 2x^2, \quad onde \;\; x = ne, \;\; se \;\; ne > ns \;\;\; ou \;\;\; x = ns, \;\; se \;\; ns > ne \tag{4.5}$$

onde, ***ne*** é o número de entradas e ***ns*** é a quantidade de saídas.

O indivíduo é enviado à função de avaliação como vetor. Este é dividido como matriz, colocando-se os valores nas células da tabela. Com cada gene no seu devido lugar, é iniciado o processo de avaliação do cromossomo como portas lógicas e suas conexões (Figura 4.3).

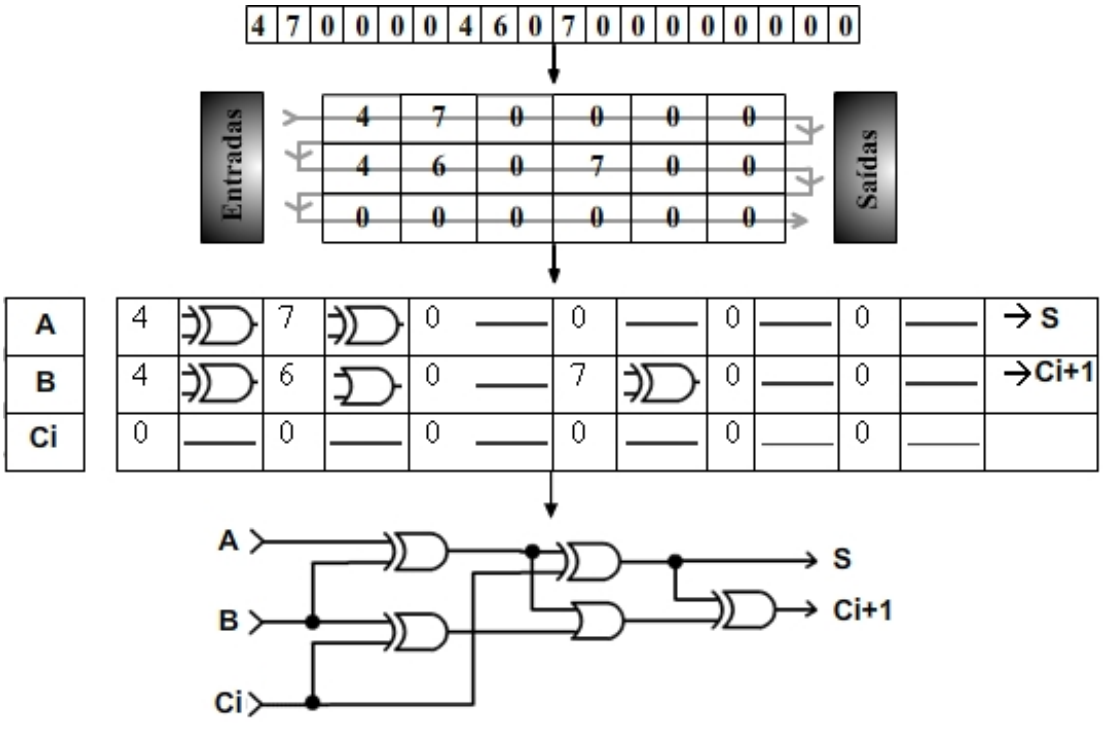



Figura 4.3 – Cromossomo gerado e circuito resultante.

O processo de avaliação é iniciado pela obtenção das primeiras entradas fornecidas pela tabela verdade, que neste caso são A=0, B=0 e Ci=0. Com esses valores

armazenados, parte-se para a simulação da porta lógica proposta. O elemento inicial é o da primeira linha e primeira coluna do cromossomo disposto em forma matricial (Figura 4.3). O valor resultante da operação logica que para este exemplo é uma porta XOR, que é armazenado e servirá como entrada para as portas lógicas nas células da segunda coluna na matriz.

A segunda conexão testada está na segunda linha e primeira coluna da tabela, a terceira está na terceira linha e primeira coluna. Depois testa-se a segunda coluna linha por linha, incrementando-se as colunas até que todo o cromossomo seja testado.

Com todas as operações lógicas realizadas, compara-se o valor obtido na célula da última coluna e primeira linha com a tabela verdade. Se os valores coincidirem, então o cromossomo corresponde ao circuito proposto pela tabela verdade para aquela entrada. No caso de múltiplas saídas, elas são dispostas conforme a significância na tabela verdade, partindo do bit de maior significância que será colocado na primeira linha e os consecutivos de menor significância nas linhas inferiores da matriz do cromossomo. Este processo é realizado para todas as entradas propostas pela tabela verdade. O circuito correto é aquele cuja as saídas correspondem a tabela verdade original proposta.

### 4.2.2 Circuito Sequencial

Um circuito combinacional é aquele em que as saídas dependem unicamente das entradas. Pode-se representar um circuito combinacional qualquer através do modelo da Figura 4.4.

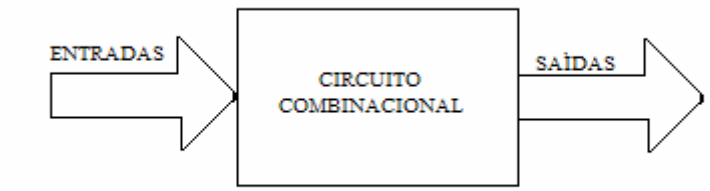



Figura 4.4 – Representação para um circuito combinacional.

Em um circuito sequencial a saída depende não somente das entradas presentes, mas também da história das entradas no passado. Possui realimentação e elememtos de

memória, para armazenar o estado anterior do circuito. Representa-se um circuito sequencial conforme figura.

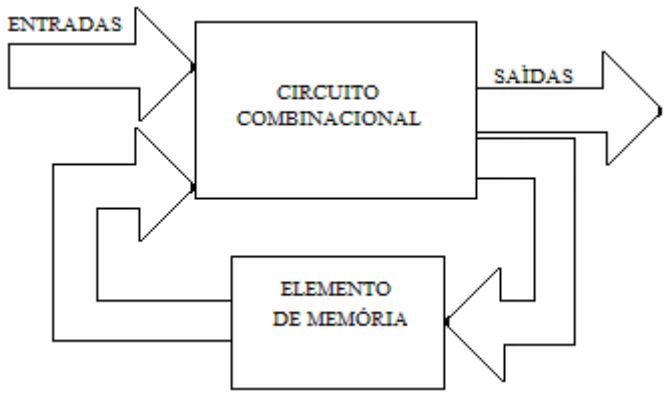



Figura 4.5 – Representação de um circuito seqüencial.

As unidades de memória utilizadas para os circuitos sequenciais são conhecidas como *flip-flops*. Cada *flip-flop* é capaz de armazenar um bit (*binary digit*). Um circuito sequencial caracteriza-se por apresentar vários estados. Em cada estado o circuito armazena uma recordação de sua história passada, de modo que possa saber o próximo passo.

Diferentemente dos circuitos combinacionais que são caracterizados por uma Tabela Verdade, e através do mapa de Karnaugh obtém-se o circuito de portas lógicas minimizado, o que descreve o comportamento dos circuitos seqüenciais são o diagrama de estados. A partir deste diagrama, obtém-se a tabela de estados, tabela de transição de estados, equações de próximos estados, de saída e *flip-flops* até a minimização através do MK, para finalmente obter-se a lógica de portas junto com os elementos de memória (*flip-flops*).

A partir destes dados foi executado um experimento em síntese de sistemas seqüenciais usando a teoria do EHW. A Figura 4.6 mostra o diagrama de um circuito seqüencial ou uma máquina de estados, que tem dois estados, o estado A e o B. Pelo diagrama de estados foi construída a tabela de transição, Tabela 4.3. Por ter dois estados, convenciona-se que A representa o bit ‘0’ e B ‘1’, e sendo assim somente um *flip-flop* é necessário para compor o circuito para esta máquina de estados. Se o número de estados estiver na faixa de, $2^{nf-1}+1$ *a* $2^{nf}$ serão necessários ***nf*** *flip-flops* para o circuito.

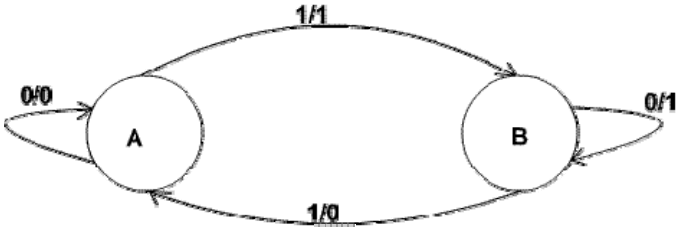



Figura 4.6 – Diagrama da máquina de estados.

Tabela 4.3 – Tabela de Estados.

| Entrada | Estado atual | Próximo estado | Saída |
|---|---|---|---|
| 0 | A | A | 0 |
| 0 | B | B | 1 |
| 1 | A | B | 1 |
| 1 | B | A | 0 |

Tabela 4.4 – Tabela de Transição.

| Entrada | Estado atual | Próximo estado | Saída |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

Tabela 4.5 – Tabela de transição do flip-flop JK.

| Estado anterior | Próximo estado | J | K |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | X | 1 |
| 1 | 1 | X | 0 |

Tabela 4.6 – Tabela fornecida ao GA.

| Entrada | | Saída | | |
|---|---|---|---|---|
| Entrada | Estado atual | Saída | J | K |
| 0 | 0 | 0 | 0 | X |
| 0 | 1 | 1 | X | 0 |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | X | 1 |

Escolhendo flip-flop JK, descrito pela tabela de transição conforme a Tabela 4.5, compõe-se a tabela final onde é mostrado todo o comportamento do circuito.

Pela Tabela 4.6 define-se o cromossomo padrão para compor a população do AG e assim iniciar o processo de busca para o circuito combinacional. Este circuito gerado, junto com os *flip-flops* resultaram em um circuito seqüencial descrito pela tabela de transição anterior. No final do processo o cromossomo encontrado gera o código VHDL (*Very High Speed Integrated Circuit Hardware Description Language).*

## 4.3 GERAÇÃO DO ARQUIVO VHDL

O VHDL é uma linguagem de programação utilizada para a simulação e síntese de circuitos estando disponível no ambiente QUARTUSII da Altera (ALTERA, 2006). A partir do melhor cromossomo encontrado durante o processo de evolução do AG e com base nos dados contidos nesse cromossomo é gerado o código em VHDL que servirá para programar um dispositivo PLD. Os testes foram realizados em PLD's da família FLEX 10K (EPF10K70) que possui 118 000 portas, 3 744 elementos lógicos e o número máximo de 358 pinos I/O (input/output). O diagrama de blocos com sua arquitetura interna são mostrados na Figura 4.7.

Na Figura 4.8 mostra-se este dispositivo montado em uma placa didática denominada UP2 – comercial do fabricante ALTERA, disponível nos laboratórios do departamento de Engenharia Elétrica da Faculdade de Engenharia de Ilha Solteira- UNESP.

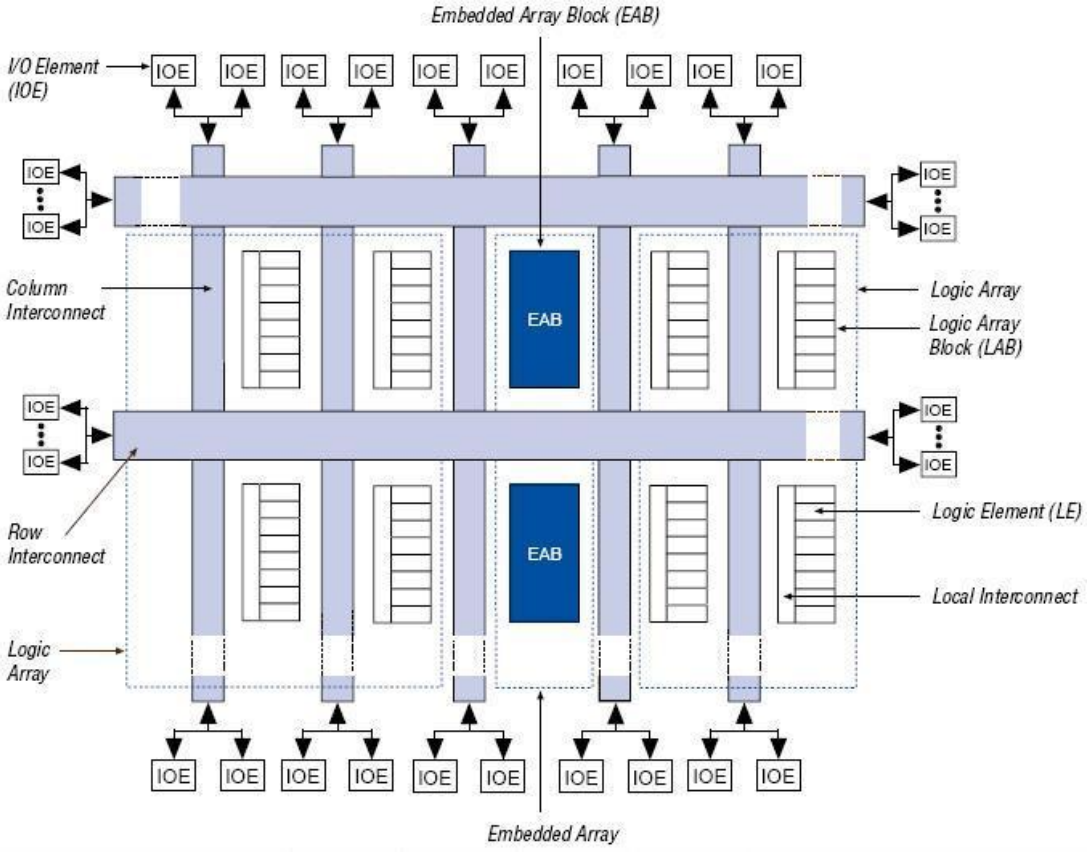



Figura 4.7 – Diagrama de blocos do dispositivo FLEX 10K (ALTERA, 2006).

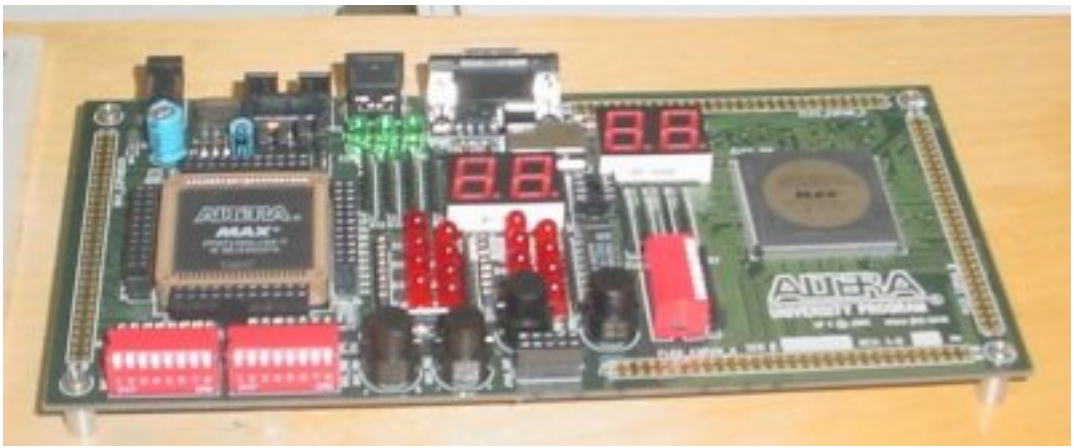


Figura 4.8 – Placa UP2 (ALTERA, 2006).

A partir do melhor cromossomo inicia-se o processo de geração do arquivo vhdl. Primeiro são criadas as entidades lógicas que fazem parte da codificação (FIO, NOT, AND, OR e XOR), depois é gerado um cabeçalho padrão que define as portas de entrada e saídas dessas entidades, em seguida o cromossomo é “quebrado” e disposto na forma de matriz.

A segunda fase equivale a conexão das portas lógicas criadas, conforme o cromossomo resultante. Cada gene do cromossomo gera uma linha de programação com o tipo de porta lógica e suas ligações. Por fim, são escritas no arquivo as linhas de programação que determinam as saídas do circuito. Todo este processo é ilustrado pelo fluxograma da Figura 4.9.

Estão descritos no apêndice A as entidades geradas para as portas lógicas definidas no trabalho e um exemplo de um arquivo para um cromossomo usando essas entidades, na linguagem VHDL.

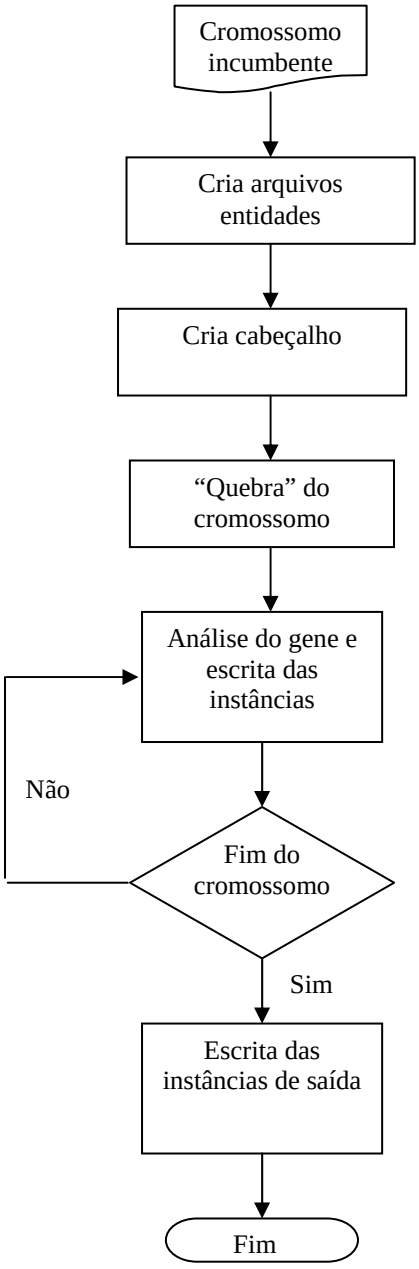



Figura 4.9 – Fluxograma que gera o código VHDL.

# 5 RESULTADOS EXPERIMENTAIS

Apresenta-se nesta seção alguns exemplos de EHW extrínsecos simulados em linguagem de programação C e linguagem de descrição de hardware VHDL. Estes exemplos foram realizados em um computador portátil da marca Compaq série R4012US com processador AMD Athlon 64 bits de 2 GHz de velocidade e 512MB de memória RAM.

Os circuitos foram implementados com o objetivo de avaliar o desempenho do EHW a partir do algoritmo genético proposto.

Os exemplos implementados usaram duas representações diferentes isto é, a representação do circuito por mapas de fusíveis e codificação binária, e a representação por portas lógicas matricial binária e numérica.

Na representação por mapas de fusíveis foram implementados somente circuitos combinacionais, tais como um somador completo e um detector de números primos de 4 bits.

Para a representação com portas lógicas foram realizados a síntese de circuitos combinacionais de funções lógicas simples e um somador binário completo. Usando esta mesma representação foram implementados exemplos de circuitos seqüenciais, e neste trabalho são descritos dois exemplos de máquina de estados com dois e cinco estados. Os circuitos gerados pelo EHW foram testados no ambiente do Software Quartus II da *Altera Corporation* conforme já detalhado no capítulo 4.

Em todas as simulações realizadas, utilizou-se uma representação do tipo fixa e os operadores genéticos seleção, recombinação e mutação já especificados em capítulos anteriores. Com este algoritmo proposto, verifica-se uma convergência rápida e uma menor quantidade de iterações.

O algoritmo proposto é iterativo e portanto o usuário deve fornecer alguns parâmetros necessários para o inicio e a continuidade de execução do programa, tais como:

- Nome do arquivo texto que contém a descrição do circuito em questão:
- Quantidades de entradas e saídas, tamanho da população, número de iterações;
- Taxa de recombinação e a taxa de mutação.

Os arquivos de entrada do programa contêm uma tabela que descreve o comportamento dos circuitos a serem simulados. A Figura 5.1 ilustra os detalhes destes arquivos.

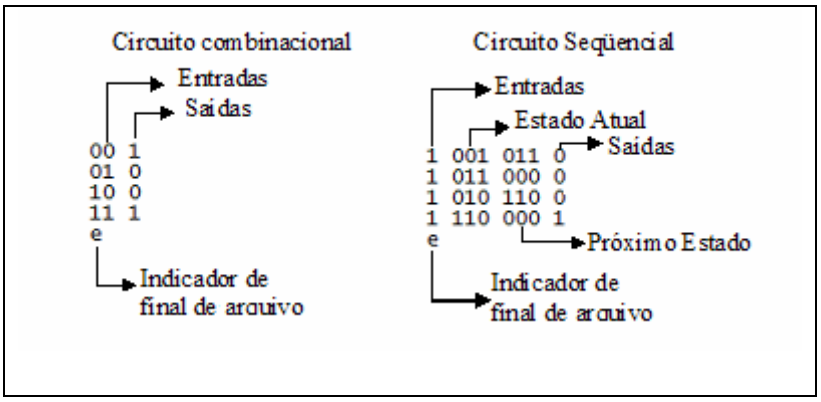



Figura 5.1 – Detalhes do arquivo de entrada.

## 5.1 REPRESENTAÇÃO USANDO MAPAS DE FUSÍVEIS.

*Somador binário*

O Somador binário é um circuito combinacional que realiza a soma completa de uma coluna de números binários. Sua tabela verdade é expressa através da Tabela 5.1.

Tabela 5.1 - Tabela verdade do somador implementado.

| Entradas | | | Saídas | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Inicialmente, implementou-se o somador binário completo que tem 3 entradas e 2 saídas conforme mostra a representação em mapas de fusíveis na Figura 5.8. Conforme o mapeamento de fusíveis o somador possui 3 entradas (3 inversores), 7 portas AND’s e 2

saídas obtendo conforme a equação (2.1) no capítulo 2, 56 genes. Como são 8 linhas que definem a tabela verdade, os acertos devem totalizar 8 pontos.

Com o objetivo de redução do esforço computacional na simulação, omite-se na representação e portanto, no cromossomo, o caso de entradas zero da tabela verdade que resulta em saída zero.

Para as simulações do algoritmo genético, considera-se uma população de 250 indivíduos gerados aleatoriamente. Depois de várias execuções o algoritmo foi calibrado para uma taxa de recombinação de 80% e uma taxa de mutação em 10%, sempre com 50 iterações. É possível encontrar soluções válidas com outros valores, entretanto estes foram os que resultaram na maior incidência de cromossomos corretos.

Os resultados das simulações são mostrados na Figura 5.3. Na Figura 5.4 foram usada taxas de recombinação e mutação diferentes como 70% e 1%, na tentativa de avaliação do comportamento e convergência do algoritmo. Vê-se que para este último caso, não foi gerado um cromossomo correto, ou seja, houve uma saturação da população por falta de diversidade.

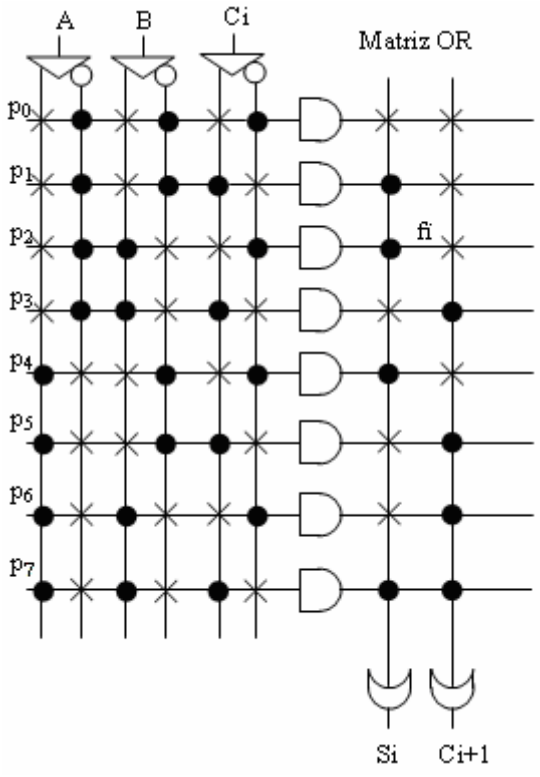



Figura 5.2 – Mapa de fusíveis do somador

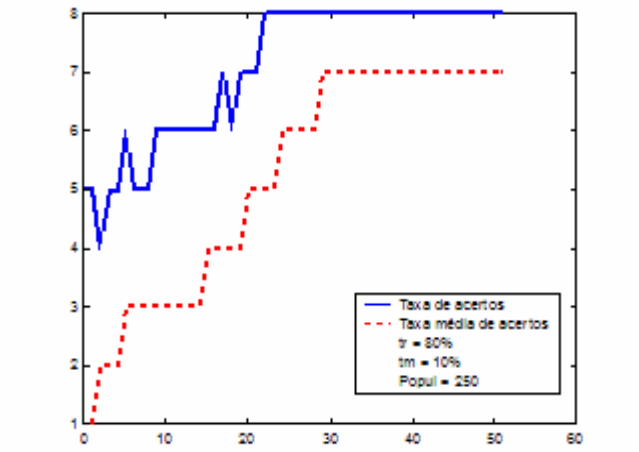



Figura 5.3 – Número de iterações x Acertos – Houve convergência.

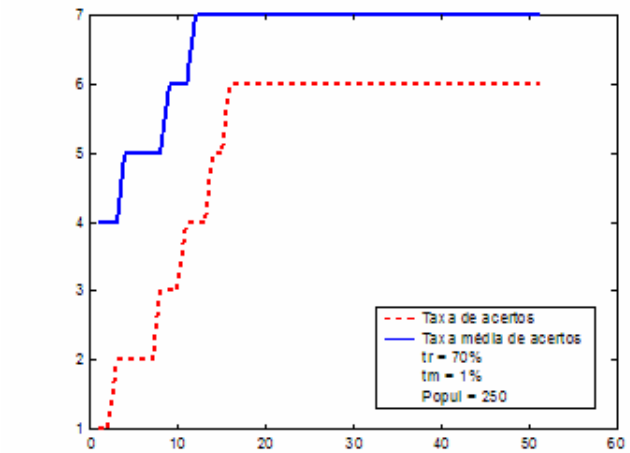



Figura 5.4 – Número de iterações x Acertos – Convergência para resultado errado.

Depois de obtido o resultado, usa-se o mesmo ambiente Quartus II-ALTERA para realizar uma simulação (Figura 5.5) e comprovar a síntese do circuito somador completo através do algoritmo evolutivo.

Os cromossomos e suas representações em mapas de fusíveis podem ser observados na Figura 5.6 (cromossomo correto) e Figura 5.7 (cromossomo incorreto).

As entradas no mapa correspondem as variáveis A, B e Ci e as saídas são Si e Ci+1.

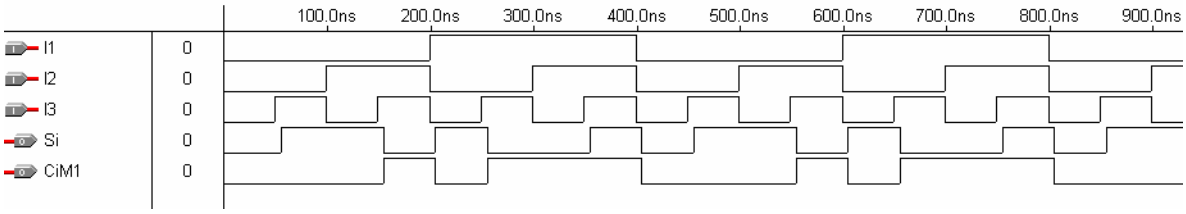



Figura 5.5 – Simulação para o cromossomo correto.

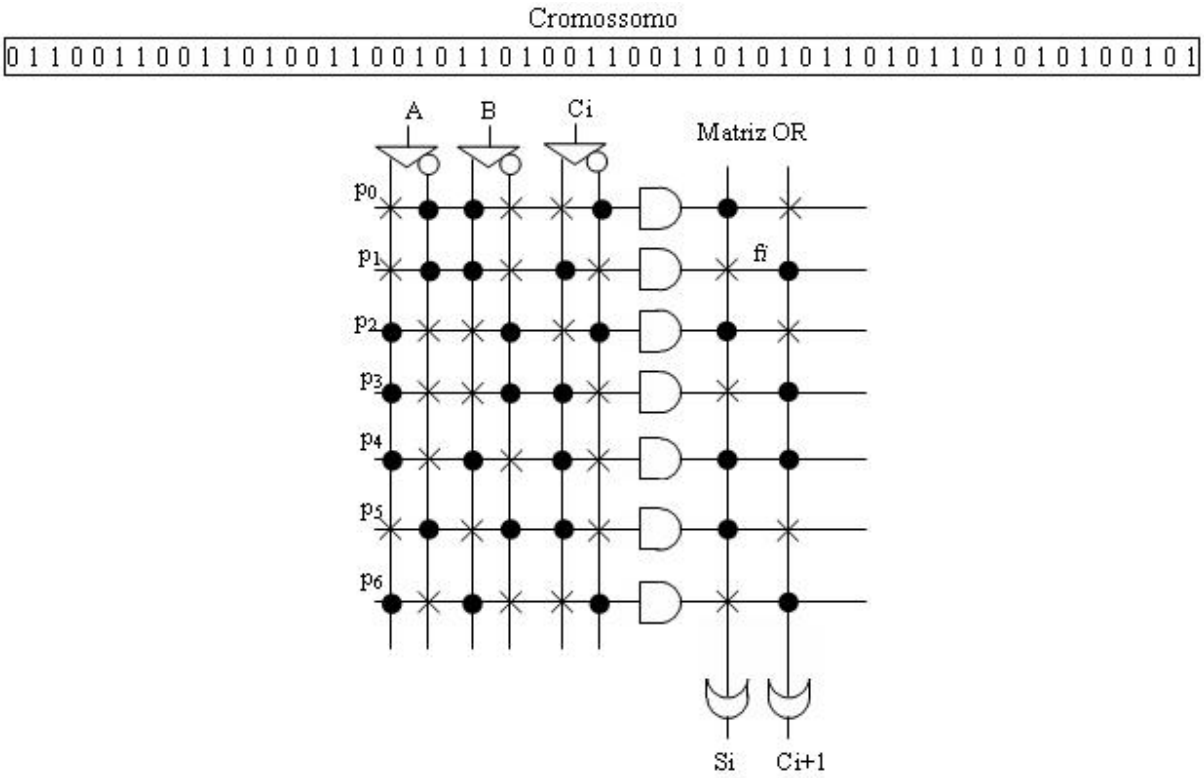



Figura 5.6 – Cromossomo e mapa de fusíveis correto.

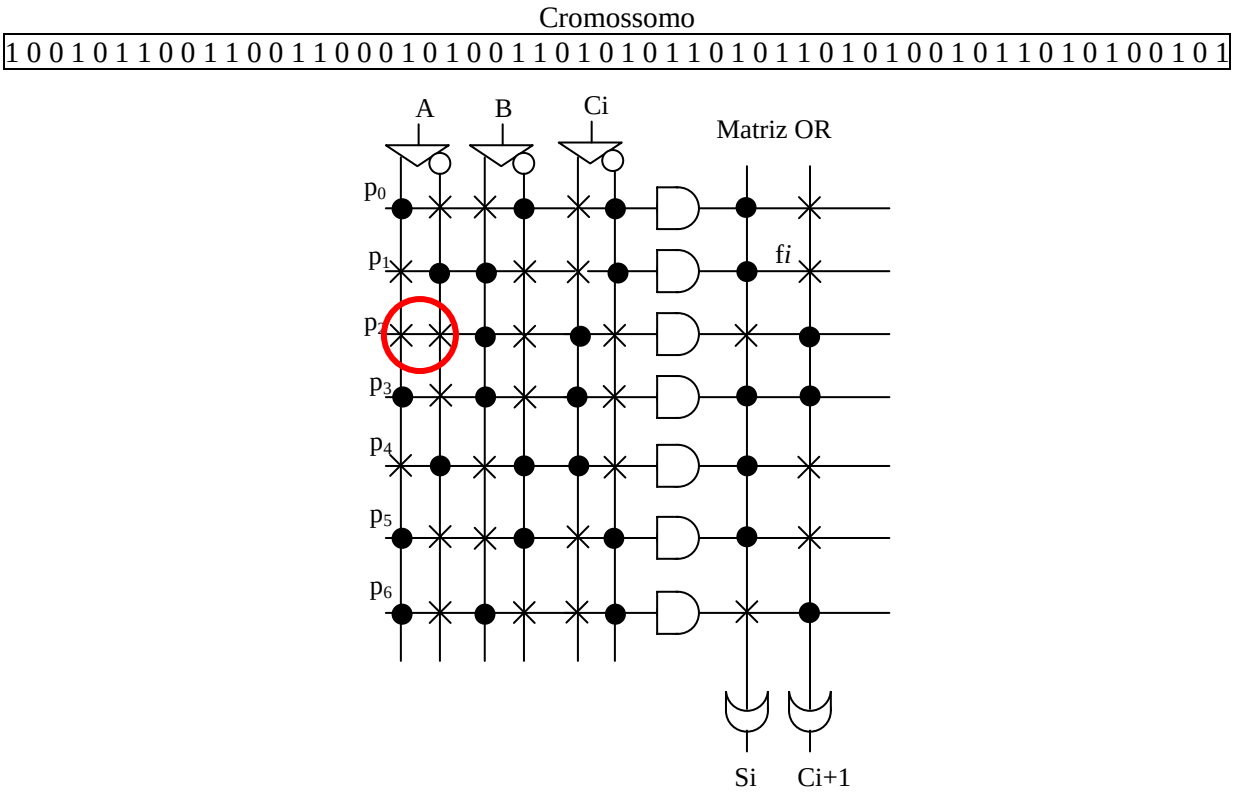



Figura 5.7 – Cromossomo e mapa de fusíveis incorretos.

*Detector de números primos de 4 bits*

Este circuito combinacional exibe uma saída ativa quando sua entrada binária corresponde a um número primo decimal, ou seja, números que são divisíveis por ele mesmo ou pela unidade, excluído o 1. Portanto, usando uma representação de 4 bits têm-se os números primos 2,3,5,7,11,13, conforme tabela verdade Tabela 5.2.

Tabela 5.2 – Tabela Verdade para o detector de números primos.

| Entradas | | | | Saída |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Para este exemplo com 4 entradas e uma saída, o mapa de fusíveis foi montado conforme a Figura 5.8. Para a determinação da quantidade de genes do cromossomo utilizou-se a equação 2.1 no capítulo 2 e obteve-se a quantidade de 54 genes. Para simplificar as buscas no algoritmo, as saídas “0” foram abolidas da representação. Conforme a tabela verdade para achar o circuito correto ou cromossomo o acerto ou função objetivo seja igual a 16.

O programa criado para sintetizar esse circuito foi testado diversas vezes até se obter um resultado satisfatório para o cromossomo. Devido ao tamanho do cromossomo, foi

usado no algoritmo uma população maior, permitindo uma maior diversidade na população e taxas elevadas de recombinação e mutação.

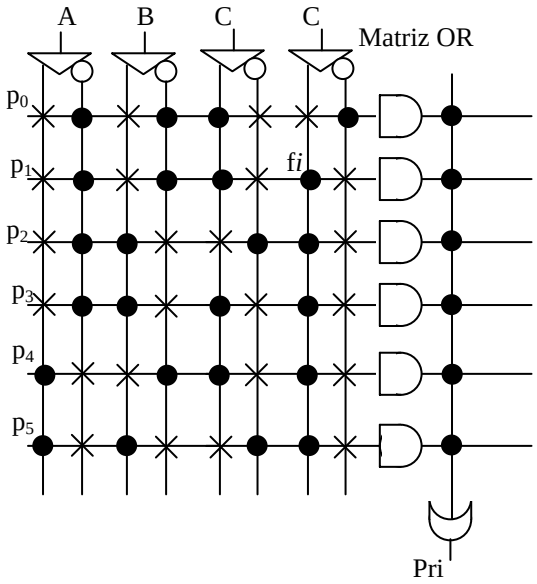



Figura 5.8 – Mapa de fusíveis para o detector de números primos.

Com 250 indivíduos, as iterações foram mantidas em 50, a taxa de recombinação foi de 80% e a de mutação foi 20%. A Figura 5.9 mostra a evolução do AG com essas configurações. Para seguir com as comparações das simulações anteriores, manteve-se a taxa de recombinação em 70% e a taxa de mutação em 1%, obtendo 15 pontos dos 16 necessários. Esses parâmetros forneceram uma simulação ilustrada na Figura 5.10.

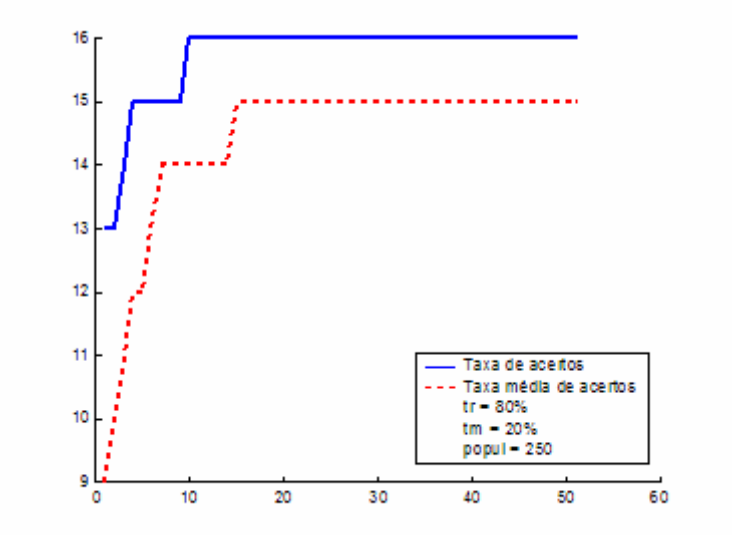



Figura 5.9 – Desempenho do AG x iterações.

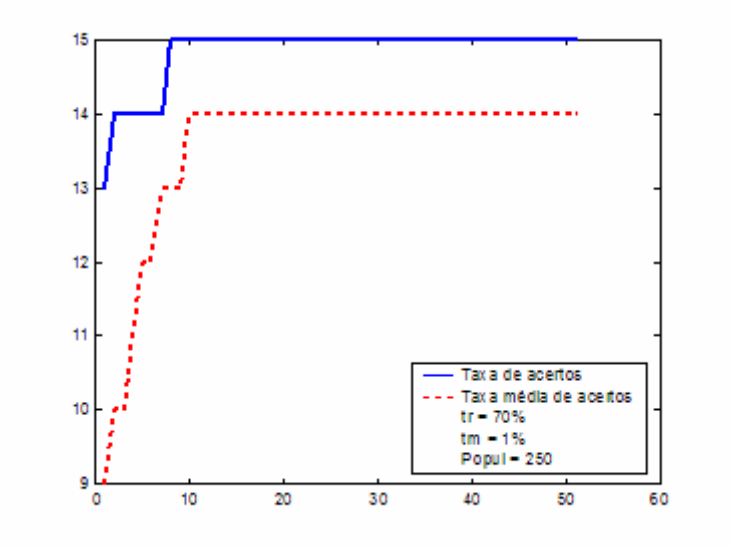



Figura 5.10 – Desempenho do AG x iterações.

A validação para este programa também é testada através do software Quartus II e pode-se observar na Figura 5.11 a sua simulação em forma de ondas. A simulação mostrada na Figura 5.12 exibe a simulação em formas de ondas do cromossomo incorreto e por conseqüência sua simulação não exibe as configurações especificadas na tabela verdade corretamente.

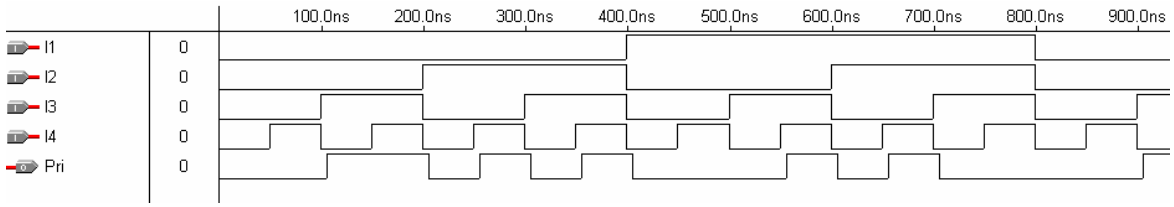



Figura 5.11 – Simulação para o cromossomo correto.

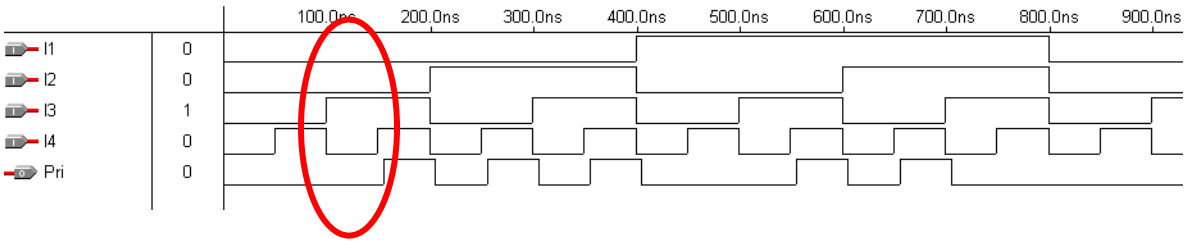



Figura 5.12 – Simulação para o cromossomo incorreto.

O cromossomo correto é exibido na Figura 5.13, juntamente com a sua representação em mapas de fusíveis.

Cromossomo

0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1

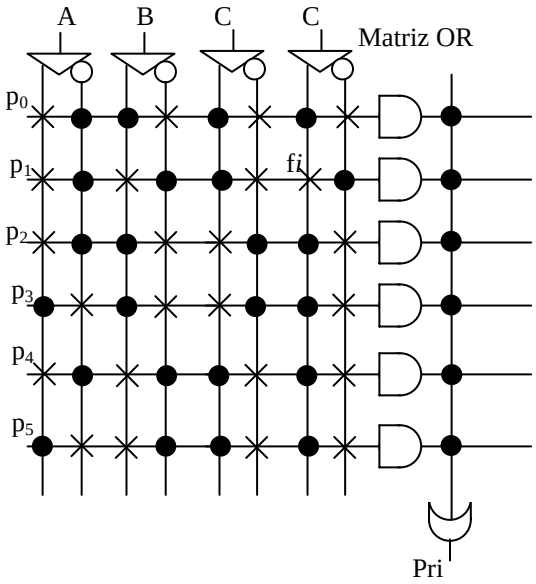



Figura 5.13 – Cromossomo e mapa de fusíveis correto.

A Figura 5.12 mostrou a simulação em forma de ondas do cromossomo incorreto apresentado na Figura 5.14. Foi possível observar neste caso uma maior dificuldade de afinação do AG, por sua necessidade de diversidade.

Cromossomo

1 0 0 1 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1

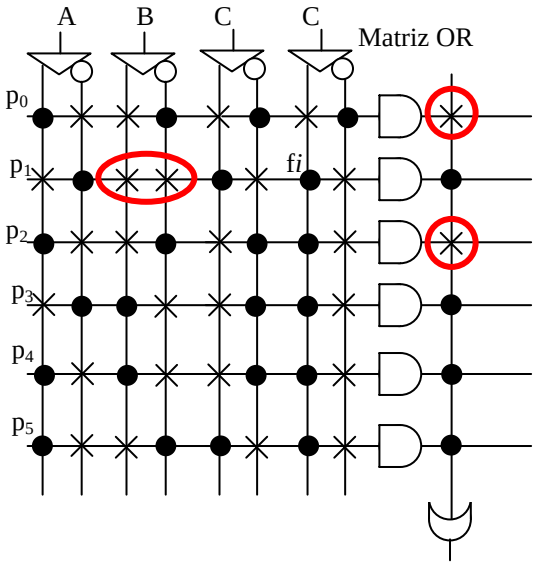



Figura 5.14 – Cromossomo e mapa de fusíveis incorreto.

## 5.2 PORTAS LÓGICAS COM CODIFICAÇÃO BINÁRIA

*Circuito combinacionais*

O primeiro exemplo implementado foi um circuito combinacional de três entradas e uma saída, cuja tabela verdade é mostrada na Tabela 5.3 e a Figura 5.15 mostra a evolução do AG. Os parâmetros utilizados na evolução do AG foram:

- $N^{\underline{0}}$ da população igual a 500 indivíduos;
- $N^{\underline{0}}$ de iterações, 200 totalizando portanto, 100.000 análises;
- Taxa de recombinação, 80% e de mutação 40%;
- Tamanho do cromossomo igual a 72 genes.

Tabela 5.3 – Tabela verdade.

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

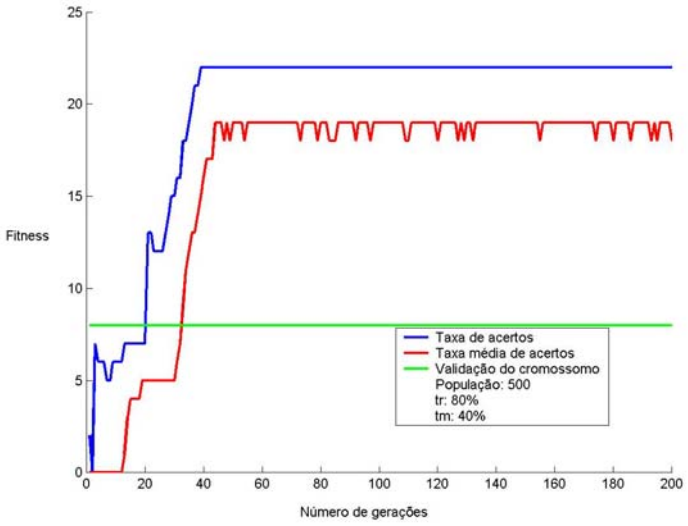



Figura 5.15 – Desempenho do AG x Iteração.

A linha reta no *fitness* igual a 8 (número de acertos) determina a validação do cromossomo. Compara-se os resultados obtidos neste trabalho para a mesma função, com os resultados de (COELLO et al., 2001), que usa uma codificação de números inteiros, e os resultados obtidos por mapa de Karnaugh, apresentados na Tabela 5.4.

Tabela 5.4 – Comparação dos resultados.

| **(COELLO et al., 2001)** | **Mapa de Karnaugh** | **Programa proposto** |
|---|---|---|
| F = C (A + B) $\oplus$ (A B) | F = C ( A $\oplus$ B) + B (A $\oplus$ C) | F = [A' $\oplus$ (B' $\oplus$ C') ]. (B + C) |
| 4 gates | 5 gates | 4 gates |
| 2 AND's, 1 OR, 1 XOR | 2 AND's, 1 OR, 2 XOR's | 2 XOR's, 1 AND, 1 OR |
| 700indivíduos, 400geraçoes, $t_r$ =50% e $t_m$ =0,7% | | 500 indivíduos , 200 gerações, $t_r$ =80% e $t_m$ =40% |

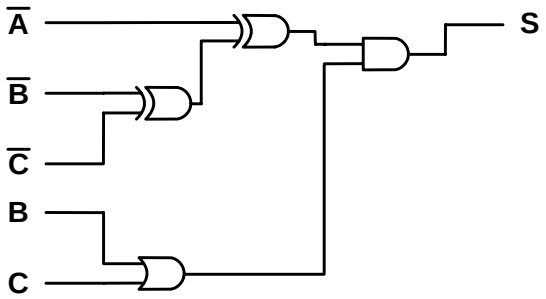



Figura 5.16 – Circuito resultante do programa implementado.

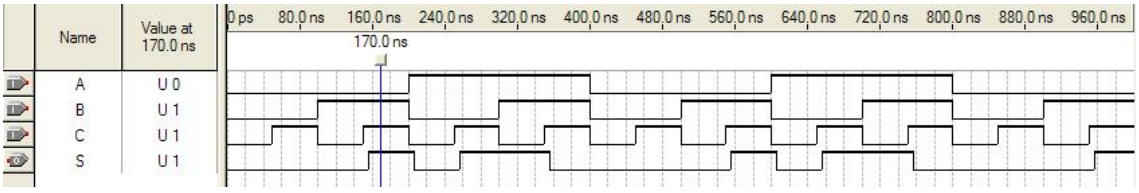



Figura 5.17 – Simulação para o cromossomo correto.

O segundo exemplo de síntese foi realizado para um circuito combinacional de quatro entradas e uma saída, descrito pela tabela verdade, Tabela 5.5, portanto definindo uma função de avaliação de 16 acertos. A representação desse circuito resultou em um cromossomo binário de 160 genes. Os resultados obtidos se encontram na Tabela 5.6, onde são mostradas também as taxas e as restrições utilizadas. Vale ressaltar, que foram necessárias várias execuções do programa para alcançar a convergência do algoritmo.

Tabela 5.5 – Tabela verdade,

para o segundo exemplo.

| A | B | C | D | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Tabela 5.6: Comparação dos resultados.

| **COELLO et al., 2001** | **Mapa de Karnaugh** | **Programa proposto** |
|---|---|---|
| F = {A ⊕ [(B ⊕ D)+CD]} ⊕ [A + (C + D)]' | F = [(A'C) ⊕ (D'B')] + [(C'D)(C ⊕ B')] | F = [A . (C + D)]' ⊕ [(B' ⊕ C') (C' + D')] |
| 8 gates | 11 gates | 8 gates |
| 1 AND, 3 OR's, 3 XOR's, 1NOT | 4 AND's, 1 OR, 2 XOR's, 4 NOT's | 2 AND's, 2 OR's, 2 XOR's, 2NOT's |
| 2 000 indivíduos, 400gerações $t_r$ =50% e $t_m$ = 0,22% | | 500 indivíduos, 500 iterações $t_r$ =80% e $t_m$ =40% |

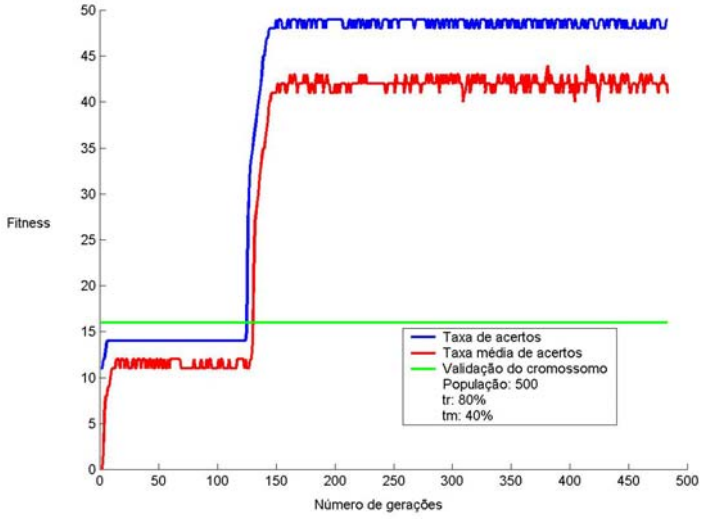



Figura 5.18 – Desempenho do AG x Iteração.

O circuito obtido pelo melhor cromossomo (incumbente) é exposto na Figura 5.19 e sua simulação através do programa Quartus II é mostrada pela Figura 5.20 onde se observa o correto funcionamento do circuito.

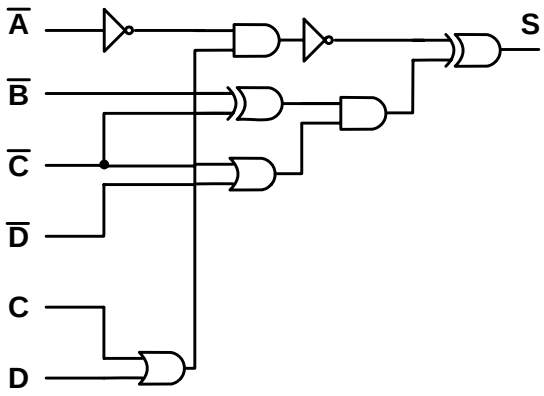



Figura 5.19 – Circuito gerado pelo EHW.

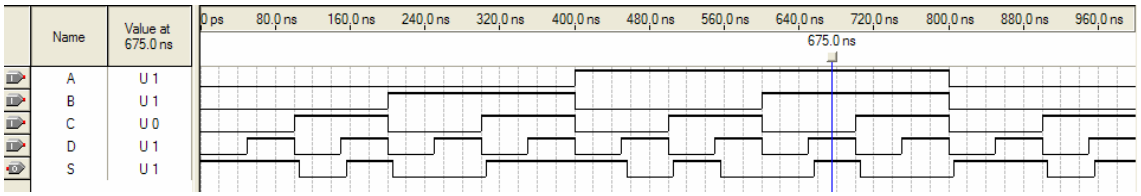



Figura 5.20 – Formas de ondas resultantes da simulação.

## 5.3 PORTAS LÓGICAS COM CODIFICAÇÃO INTEIRA

*Circuitos combinacionais*

O primeiro exemplo consiste em um circuito combinacional de três entradas e uma saída, dado pela expressão booleana $F_1$ :

$$F_1 = \sum_{ABC}(3,5,6)$$

Portanto, para três variáveis na função, a função objetivo deve totalizar um número de acertos igual a 8. Baseado na teoria mencionada no capítulo 4, compõem uma matriz de ordem 3x3 que determina a quantidade de genes, ou seja, tamanho do cromossomo (string numérico) de 9 genes.

Os parâmetros de controle usados para convergência do algoritmo foram:

- tamanho da população, 500;
- número máximo de iterações, 500;
- taxa de recombinação, 50% e mutação, 5 %.

EHW: $F_1 = A(B+C) \oplus (B \oplus C)$

4 portas lógicas: 2 XOR's, 1 OR, 1 AND.

Mapa de Karnaugh: $F_1 = C(A \oplus B) + B(A \oplus C)$

5 portas lógicas: 2 AND's, 1 OR, 2 XOR's.

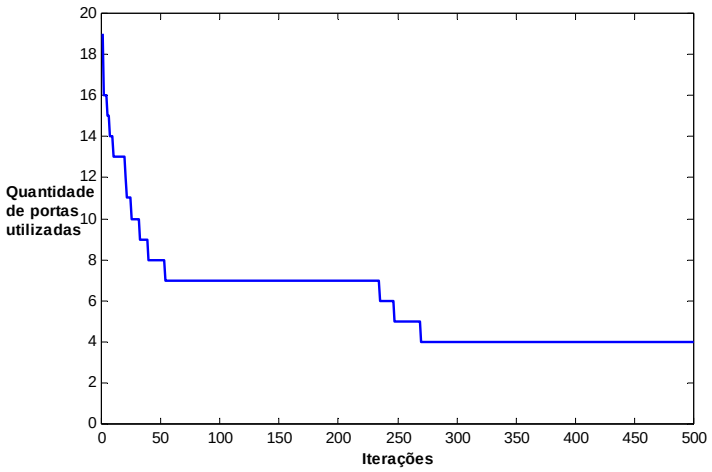



Figura 5.21 – Quantidade de portas lógicas x Iteração.

Observa-se neste gráfico da Figura 5.21 a ordem decrescente da linha. Esta mudança é efetuada para enfatizar a redução da quantidade de portas lógicas utilizadas (minimização) para representar o circuito que neste exemplo iniciou com 18 portas lógicas e terminou com 4.

*Circuito de quatro entradas e uma saída*

O segundo exemplo é um circuito combinacional de quatro entradas e uma saída:

$F_2 = \sum_{ABCD}(0,1,3,6,7,8,10,13)$

O cromossomo tem 32 genes definindo uma matriz bidimensional de 4x8. População inicial aleatória, tamanho da população, 1000; número de iterações, 1000; taxa de recombinação, 50 %; taxa de mutação, 2 %;

EHW:

$$F_2 = \left\{A\left[A(C+D)\right] \oplus (B \oplus C)(CD)'\right\}'$$

9 portas lógicas: 4 AND's, 2 XOR's, 2 NOT's, 1 OR.

Mapa de Karnaugh:

$$F_2 = \left[(A'C) \oplus (D'B')\right] + \left[(C'D)(C \oplus B')\right]$$

11 portas lógicas: 4 AND's, 4 NOT's, 2 XOR's, 1 OR.

Na Figura 5.22, observa-se o desempenho do AG para este segundo exemplo, com a obtenção do circuito minimizado, aproximadamente na geração 990.

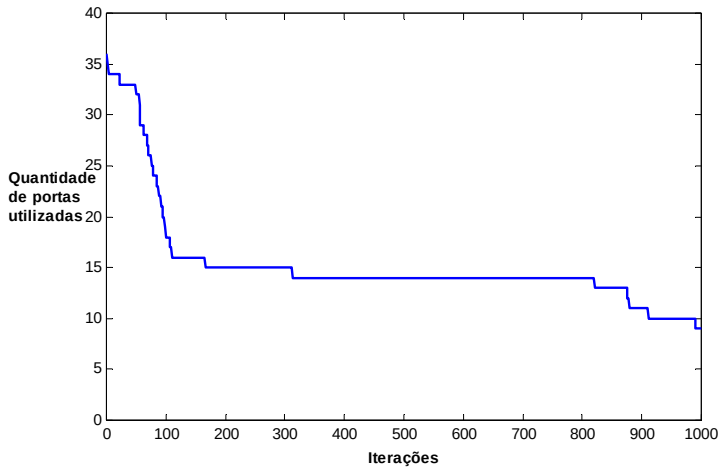



Figura 5.22 – Quantidade de portas utilizadas x Iteração.

*Somador binário completo*

Neste exemplo, toma-se um somador completo (circuito combinacional), que realiza a soma entre dois números binários (A e B) de um bit cada um (conforme Tabela 5.7). Comparam-se os resultados obtidos através da minimização usando Mapa de Karnaugh (Figura 5.23), com os resultados obtidos pelo EHW Figura 5.24. O resultado obtido pelo mapa de Karnaugh, $S = A \oplus B \oplus C_i$ e $C_{i+1} = BC_i + AC_i + AB$, onde $C_i$ representa o vai um de entrada e $C_{i+1}$ o vai um de saída.

Neste exemplo l, baseado na proposta do circuito evolutivo, o cromossomo foi mantido com 18 genes definindo uma matriz bidimensional de 3x6. População inicial aleatória, tamanho da população, 1000; número de iterações, 500; taxa de recombinação, 50%; taxa de mutação, 3%.

Tabela 5.7 – Tabela Verdade para o somador.

| A | B | $C_i$ | S | $C_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

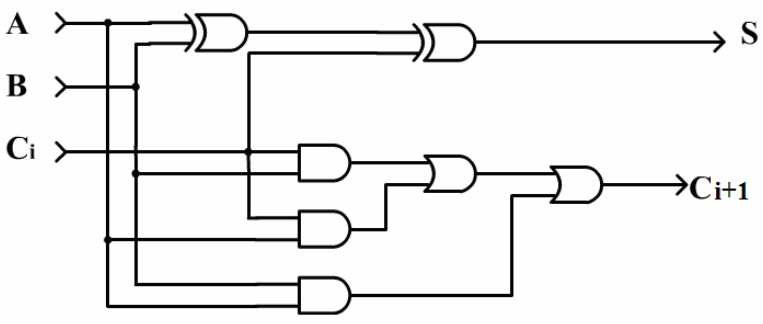
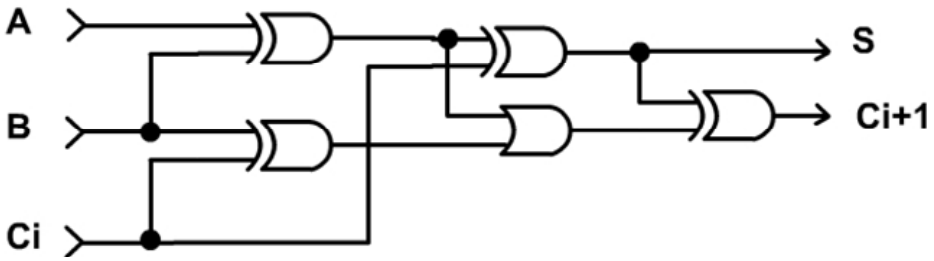



Figura 5.23 Circuito somador minimizado pelo Mapa de Karnaugh.

Na Figura 5.24 (a) e (b), mostra-se o cromossomo gerado e o circuito obtido, respectivamente. Observa-se o desempenho do AG na Figura 5.25 e a sua convergência para o resultado esperado na geração 100, aproximadamente.

| 0 | 4 | 0 | 0 | 0 | 4 | 5 | 0 | 0 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

a. Cromossomo gerado.




b. Circuito obtido.

Figura 5.24 – Somador gerado pelo EHW.

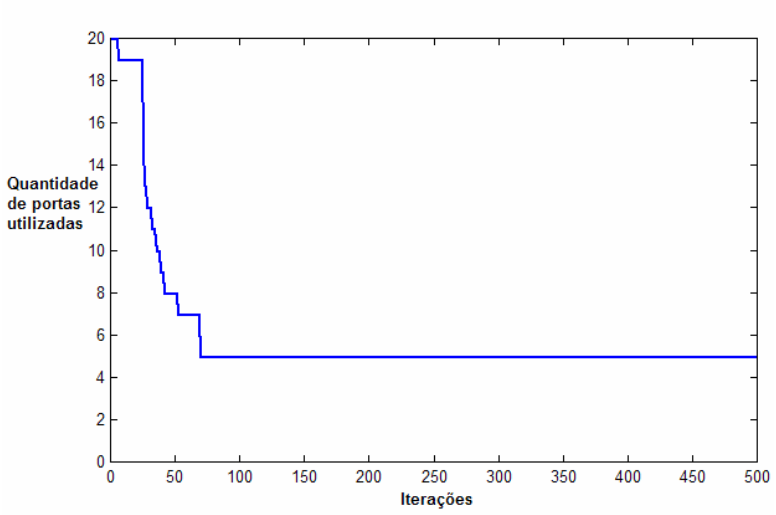



Figura 5.25 – Convergência do EHW

## 5.4 CIRCUITOS SEQÜENCIAIS

Foram implementados dois exemplos de circuitos lógicos seqüenciais para validar o algoritmo apresentado no capítulo 4. Nesta simulação busca-se o circuito combinacional que junto com os *flip-flops* definidos previamente irão compor as máquinas de estados em questão.

*Máquina com dois estados*

Primeiramente, foi implementada uma máquina com dois estados conforme o diagrama de estados e tabela de estados mostrado na Figura 4.6 e tabela 4.3, respectivamente, no capítulo 4. A convergência do AG para este exemplo é apresentada na Figura 5.26.

Para este exemplo os parâmetros utilizados foram:

- 250 indivíduos e 100 iterações;
- 50% de taxa de recombinação e 3% de taxa de mutação;

O melhor resultado foi obtido na geração 96, com a utilização de 4 portas lógicas, e para 10 execuções.

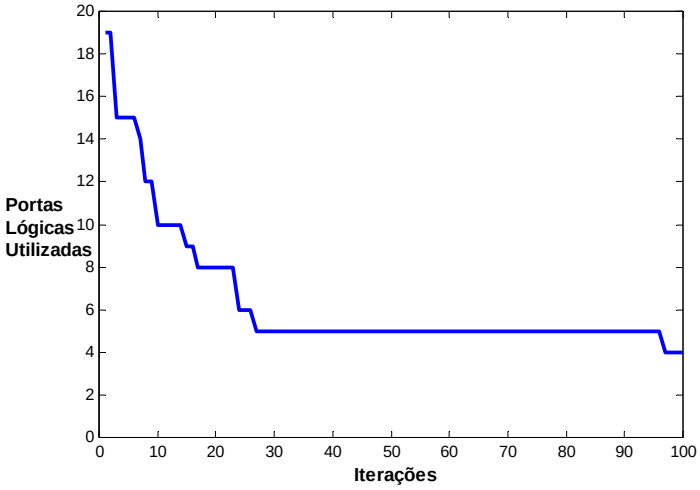



Figura 5.26 – Convergência da máquina de estados com dois estados.

*Máquina de estados com cinco estados*

O circuito seqüencial testado tem seu comportamento descrito pela Figura 5.27 e tabela de transição (Tabela 5.8). Os estados são representados por três bits sendo “000” para $S_0$, “001” para $S_1$, “011” em $S_2$, “010” em $S_3$, “110” para $S_4$. Como são necessários 3 bits para armazenar o estado em que se encontra a máquina, o sistema utiliza 3 *flip-flops*.

O cromossomo para este circuito seqüencial é constituído de 98 genes e os melhores resultados foram obtidos com 2000 indivíduos, 1000 iterações, 45% de taxa de recombinação e 1% de taxa de mutação. A convergência do circuito foi obtida em 6 de 10

execuções. Pequenas mudanças nos operadores como a implementação de técnicas diferenciadas de elitismo ou artifícios para efetuar a recombinação, podem ser realizadas para obter-se um melhor aproveitamento.

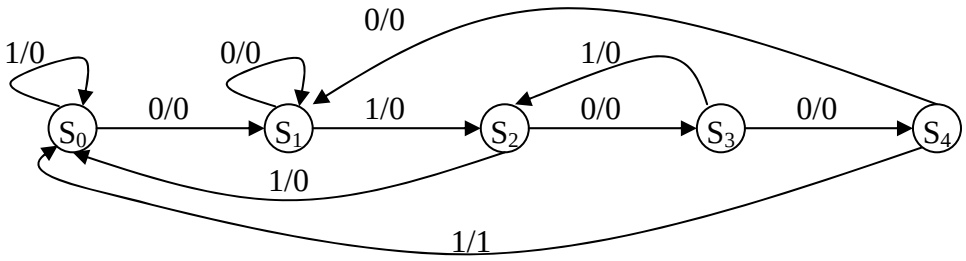



Figura 5.27 – Diagrama da máquina de cinco estados.

Tabela 5.8 – Tabela de transição de estados.

| Entrada | Estado Atual | Próximo Estado | Saída |
|---|---|---|---|
| 0 | 000 | 001 | 0 |
| 0 | 001 | 001 | 0 |
| 0 | 011 | 010 | 0 |
| 0 | 010 | 110 | 0 |
| 0 | 110 | 001 | 0 |
| 1 | 000 | 000 | 0 |
| 1 | 001 | 011 | 0 |
| 1 | 011 | 000 | 0 |
| 1 | 010 | 110 | 0 |
| 1 | 110 | 000 | 1 |

A convergência do algoritmo é ilustrada na Figura 5.28, com o melhor resultado ou o cromossomo incumbente encontrado na iteração 926, e o EHW encontrou um total de 45 portas lógicas.

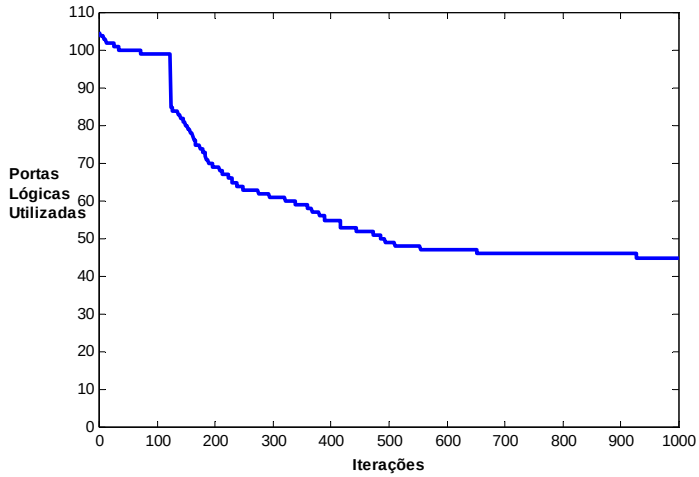



Figura 5.28 – Convergência do EHW para a máquina de cinco estados.

# 6 CONCLUSÃO

No desenvolvimento desse projeto foi aplicada a teoria EHW para executar a síntese de circuitos digitais, usando formas distintas de representação. Inicialmente utilizou-se a representação por mapas de fuzíveis com uma codificação binária. A sua aplicação apresentou resultados eficientes na síntese do circuito proposto. Com o objetivo de realizar a síntese e otimização dos circuitos foi utilizado a represesentação por portas lógicas com codificação numérica. A codificação com números inteiros reduziu o espaço de busca e o esforço computacional. Esta representação e as mudanças realizadas nos operadores que compõem o GA resultaram em uma população mais diversificada facilitando o bom desempenho e a convergência mais rápida do algorítmo.

Implementou-se para a realização dos testes com o algoritmo, circuitos combinacionais e sequenciais simples. Os circuitos sequenciais são mais complexos pois exigem além de um conjunto de circuitos combinacionais, a utilização de unidades de memória (flip-flops), porém foi usado o mesmo tipo de representação isto é, portas lógicas e a codificação inteira. Nos testes realizados com circuitos seqüenciais foram utilizados dois exemplos, uma máquina de dois estados e uma de cinco estados. Pelos resultados, observou-se uma convergência rápida, do Algoritmo Genético em ambos os casos (2 e 40 segundos) apesar da máquina de cinco estados ter um cromossomo maior (90 genes) e ser trabalhada uma população maior.

Os resultados dos exemplos simulados mostram a confiabilidade da metodologia aplicada com boa repititividade, e indicando que testes com circuitos mais complexos possam ser executados.

Foram implementados poucos exemplos com circuitos sequenciais. Por isso entende-se que seriam necessários mais alguns exemplos para que se obtivesse a determinação exata de alguns operadores.

Uma vantagen do processo evolutivo, sobre os outros processos para obtenção de síntese de projeto de circuitos digitais, é a possibilidade de usá-lo em circuitos complexos.

Contudo, observa-se que para pequenos projetos, os métodos clássicos são mais competitivos.

Outra vantagem é que na abordagem do processo evolutivo não é necessário um conhecimento a priori do circuito que se deseja projetar - guia-se por uma tabela verdade; podendo ser usado em domínios onde existem poucos conhecimentos , ou são difíceis de obter. Na abordagem convencional é necessário uma maior especificação, como número de portas , tipos e quantidade de entradas para as portas.

No projeto evolutivo pode haver vários graus de restrições, se necessário incorpora-se estes aspectos na representação cromossômica e na função de fitness.

## 6.1 Sugestões para trabalhos futuros

Como sugestões para futuros trabalhos, fundamentado nos resultados obtidos na aplicação da técnica desenvolvida, têm-se:

- Usar este algoritmo para a criação de um software educacional de síntese de circuitos digitais. Para isto, é necessário fazer uma interface amigável utilizando linguagens direcionadas para objetos;
- Testar outros tipos de técnicas para os operadores genéticos ou outros algoritmos evolutivos;
- Obtenção automática dos parâmetros de controle com base na tabela verdade inserida. Estes parâmetros podem ser calculados através de técnicas inteligentes como Redes Neurais.

# REFERÊNCIAS

ALTERA Corporation. **Introduction to quartus II**. Disponível em: http://www.altera.com/literature/manual/intro_to_quartus2.pdf. Acesso em: 24 nov. 2006.

CAMPOS, T. J.; OLIVEIRA, J. R.; JUNGBECH M. **Programação genética aplicada ao desenvolvimento de hardware evolutivo.** Disponível em: <http://congressos.eletro.ufrgs.br/cba2004/store/CD/cba2004/pdf/1279.pdf>. Acesso em: 25 jul. 2005.

COELLO COELLO, C. A.; CHRISTIANSEN, A. D.; AGUIRRE A. H. **Towards automated evolutionary design of combinational circuits, computers and electrical engineering**, Disponível em: <http://citeseer.ist.psu.edu/cache/papers/cs/8753/http:zszzszwww.lania.mxzsz%7eccoellozsztechreportszszcircuits.pdf/coello01towards.pdf>. Acesso em: 22 jun. 2007.

GOLDBERG D. E. **Genetic algorithms in search, optimization and machine learning**. Massachusetts: Addison-Wesley Publishing Company, Inc., 1989.

GOULART SOBRINHO, E. F.; MANTOVANI, S. C. A. EHW aplicado à síntese de circuitos digitais usando representação por portas lógicas. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL-SOBRAPO, 38, 2006, Goiânia. **Anais...** Goiânia: Universidade Católica de Goiás, 2006. CD-Rom.

GOULART SOBRINHO, E. F.; MANTOVANI, S. C. A. Hardware evolutivo aplicado à síntese de circuitos digitais. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 16, 2006, Salvador. **Anais…** Salvador: SBA, 2006. p. 2730-2735.

HEREFORD, J. Robust sensor system using evolvable hardware. In: CONFERENCE ON EVOLUTION HARDWARE, 2004. **Proceedings of the …** New York: IEEE/NASA/DoD, 2004. p.161-168.

HEMMI H.; MIZOGUCHI J.; SHIMOHARA K. Development and evolution of hardware behaviors. **Lecture Notes In Computer Science**, Berlin, v.1062, p. 250 – 265, 1995.

HIGUCHI, T.; TNIWA, T.; TANAKA, H.; IBA, H.; DEGARIS, FURUYA, T. Evolvable hardware with genetic learning: a first step towards. In: INTERNATIONAL

CONFERENCE ON SIMULATION OF ADAPTIVE BEHAVIOR, 2, 1993, **Proceedings of the…** New York: IEEE, 1993. p. 417 – 424.

HIGUCHI T.; IBA H.; MANDERICK, B. Evolvable hardware. In: KITANO, H.; HENDLER, J.A. (Ed.). **Massively parallel artificial intelligence.** Menlo Park: AAAIPRESS, 1994. p. 398 – 421.

HIGUCHI, T.; IWATA, M.; KEYMEULEN, D.; SAKANASHI, H.; MURAKAWA, M.; KAJITANI, I.; TAKAHASHI, E.; TODA, K.; SALAMI, N.; KAJIHARA, N.; OTSU, N. Real-world applications of analog and digital evolvable hardware. **IEEE Transactions on Evolutionary Computation**, New York, v.3, n.3, p.220 – 235, 1999.

IBA, H.; IWATA M.; HIGUCHI T. Machine learning approach to gate level evolvable hardware. **Lecture Notes In Computer Science**, Berlin, v.1259, p. 327 – 343, 1997.

IWATA M.; KAJITANI I.; YAMADA H.; IBA H.; HIGUCHI T. A pattern recognition system using evolvable hardware. **Lecture Notes In Computer Science**, Berlin, v.1141, p.761 – 770, 1996.

IWATA M.; KAJITANI I.; LIU Y.; KAJIHARA N.; HIGUCHI T. Implementation of gate – level using evolvable hardware chip, evolvable systems: from biology to hardware. **Lecture Notes In Computer Science**, Berlin, v. 2210, p. 38 – 49, 2001.

JEWAJINDA Y.; CHONGSTITVATANA P. A cooperative approach to compact genetic algorithm for evolvable hardware. **IEEE Congress of Evolutionary Computation**, New York, v.1141,p.761-770, 2006.

KAJITANI, I.; HOSHINO, T.; IWATA, M.; HIGUCHI, T. Variable length chromosome GA for evolvable hardware: evolutionary computation. In: PROCEEDINGS OF IEEE INTERNATIONAL CONFERENCE, 1996. **Proceedings of the…** New York: IEEE, 1996. p. 443 – 447, 1996.

KAJITANI I.; HOSHINO T.; NISHIKAWA D.; YOKOI H.; NAKAYA S.; YAMAUCHI T.; INUO T.; KAJIHARA N.; IWATA M.; KEYMEULEN D.; HIGUCHI T. A gate-level EHW chip: implementing GA operations and reconfigurable hardware on a single LSI, evolvable systems: from biology to hardware. **Lecture Notes In Computer Science**, Berlin, v.1478, p. 1 – 12, 1998.

KEYMEULEN, D.; DURANTEZ, M.; KONAKA, K.; KUNIYOSHI Y.; HIGUCHI T. An evolutionary robot navigation system using a gate-level evolvable hardware, **Lecture Notes in Computer Science**, Berlin, v.1259, p. 195 – 209, 1997.

KITANO, H. Evolvable hardware with development, circuits and systems, 1996. ISCAS '96. 'Connecting the World'. 1996. **IEEE International Symposium**, New York, v.4, p. 33 – 36, 1996.

KITANO, H. Challenges of evolvable systems: analysis and future directions. **Lecture Notes In Computer Science**, Berlin, v.1259, p. 125 – 135, 1997.

KOZA, J. R. **Genetic programming: on the programming of computers by means of natural selection**. Massachusetts: The MIT Press, 1992.

KOZA, J.R.; DUNLAP, F.; BENNETT III, F.H.; KEANE, M. A.; LOHN J.; ANDRE, D. Automated synthesis of computational circuits using genetic programming. In: IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION – ICEC, Indianapolis. **Conference…** Indianápolis: IEEE, p. 447-452, 1997.

LEE, D.; BAN, C.; SIM K.; SEOK, H.; LEE K.; ZHANG, B. Behavior evolution of autonomous mobile robot using genetic programming based on evolvable hardware, systems, man, and cybernetics. **IEEE International Conference**, New York, v.5, p. 8-11, 2000.

LOUIS, S. J.; RAWLINS, G. J. E. Designer genetic algorithms: genetic algorithms in structure design. In: PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS – ICGA, San Diego, 1991. **Proceedings of the…** New York: IEEE, 1991.

MANDERICK, B.; HIGUCHI T. Evolvable hardware: an outlook. **Lecture Notes In Computer Science**, Berlin, v.1259, p.305 – 311, 1997.

MANTOVANI, S. C. A.; OLIVEIRA, J.R. Síntese de circuitos digitais por evolução de circuitos. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 36, 2004, São João Del Rey. **Anais...** São João Del Rey: SBPO, 2004. p. 1820-1831.

MANTOVANI, S. C. A. GOULART SOBRINHO, E. F. Síntese de circuitos digitais através de hardware evolutivo. In: CONGRESSO LATINO-IBEROAMERICANO DE INVERTIGACIÓN OPERATIVA, 13, 2006, Montevidéu. **Anais...** Montevidéu: CLAIO 2006. CD-Rom.

MCGEER P.; SANGHAVI J.; BRAYTON R. K.; SANGIOVANNI-VINCENTELLI A. ESPRESSO-Signature: a new exact minimizer for logic functions. In: PROCEEDINGS OF THE 30TH ACM/IEEE DESIGN AUTOMATION CONFERENCE, 30, 1993, Dallas. **Proceedings of the …** Dallas: IEEE, 1993. p. 618-624.

MICHALEWICZ, Z. **Genetic algorithms + data structures = evolution programs**. 3 ed. Heidelberg: Springer-Verlag, 1996.

SANCHES E. Field programmable gate array – fpga circuits. **Lecture Notes In Computer Science**, Berlin, v. 1062, p. 1 – 18, 1996.

STOMEO, E.; KALGANOVA, T.; LAMBERT, C. A. Novel genetic algorithm for evolvable hardware. In: CONGRESS EVOLUTIONARY COMPUTATION, 2006, New York. **Cogress …** New York: IEEE, 2006. p. 134 – 141.

SUSHIL J. L. **Genetic algorithms as a computational tool for design**. 1993. 193 f. PhD (Thesis) – Department of Computer Science, Indiana University, Indiana, 1993.

TOMASSINI, M. Evolutionary algorithms. **Lecture Notes In Computer Science**, Berlin, v.1259, p. 195–209, 1997.

THOMPSON, A. Evolving eletronic robot controllers that exploit hardware resources. **Lecture Notes In Computer Science**, Berlin, v. 929, p. 640 – 656, 1995.

THOMPSON A.; HARVEY I.; HUSBANDS P. Unconstrained evolution and hard consequences, toward evolvable hardware. **Lecture Notes In Computer Science**, Berlin, v. 1062, p.136 – 165, 1996.

TYRREL, A. M.; KROHLING, R. A.; ZHOU Y. Elutionary algorithm for the promotion of evolvable hardware. **IEE Proceeding Computation Digital Technological**, New York, v. 151.

VILELA NETO, O. P.; MASIERO, L. P.; PACHECO, M. A. C.; BARBOSA, C. R. H. Evolvable hardware applied to nanotechnology. In: ADAPTIVE HARDWARE AND SYSTEMS, 1, 2006. **Conference ...** s.l.: NASA/ESA, 2006.

VON ZUBEN, F. J.; CASTRO, L. N. Computação evolutiva: uma "nova" forma de resolver problemas. Campinas: DCA-FEEC-Unicamp, 2002. Disponível em: **<ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia707_02/topico8_02.pdf>**.
Acesso em: 14 jul. 2005.

YAO, X.; HIGUCHI, T. Promises and challenges of evolvable hardware. **Lecture Notes In Computer Science**, Berlin, v. 1259, p. 55 – 78, 1997.

ZEBULUM, R. S.; PACHECO, M. A.; VELLASCO M. Evolvable system in hardware design: taxonomy, survey and applications. **Lecture Notes In Computer Science**, Berlin, v. 1259, p. 344-358, 1997.

ZEBULUM, R. S. **Síntese de circuitos eletrônicos por computação evolutiva**. 1999. Tese (Doutorado) – Pontifica Universidade Católica do Rio de Janeiro, 1999.

# Apêndice 1

## Geração dos Arquivos VHDL

---

Para exemplificar a geração dos arquivos VHDL é utilizado um circuito combinacional simples com duas entradas e uma saída (Tabela A1.1).

Tabela A1.1 – Tabela verdade do circuito exemplo.

| Entradas | | Saídas |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

O circuito combinacional descrito pela Tabela A1.1, ilustra a criação do arquivo VHDL. O cromossomo encontrado pelo EHW para esta tabela é 7 0 1 0 0 0 0 0. Esse cromossomo possui apenas duas portas lógicas (7 e 1), duas entradas e uma saída. Assim que a função recebe o cromossomo, ela cria os arquivos entidades que realizarão as operações lógicas conforme forem solicitadas. Cada operação é um arquivo .VHD.

Depois de criadas as entidades que realizarão as operações lógicas, é criado um arquivo principal que chama as entidades e realiza as ligações conforme for solicitado pelo cromossomo. O arquivo principal é iniciado por um cabeçalho padrão.

**%% Entidade NOT**

PNOT.VHD

```
ENTITY PNOT IS
PORT(
   A :IN bit;
   B :OUT bit
   );
END PNOT;
```

```
ARCHITECTURE Funcional OF PNOT IS
BEGIN
   B<= NOT (A);
End Funcional;
```

**%% Entidade Fio de ligação**

PFIO.VHD

```
ENTITY FIO IS
PORT(
   A :IN bit;
   B :OUT bit
   );
END FIO;

ARCHITECTURE Funcional OF FIO IS
BEGIN
   B<= A;
End Funcional;
```

**%% Entidade OR**

POR.VHD

```
ENTITY POR IS
PORT(
   A :IN bit;
   B :IN bit;
   C :OUT bit
   );
END POR;

ARCHITECTURE Funcional OF POR IS
BEGIN
   C <=  A OR B;
End Funcional;
```

**%% Entidade AND**

PAND.VHD

```
ENTITY PAND IS
PORT(
   A :IN bit;
```

```
   B :IN bit;
   C :OUT bit
   );
END PAND;

ARCHITECTURE Funcional OF PAND IS
BEGIN
   C <=  A AND B;
End Funcional;
```

**%% Entidade XOR**

PXOR.VHD

```
ENTITY PXOR IS
PORT(
   A :IN bit;
   B :IN bit;
   C :OUT bit
   );
END PXOR;

ARCHITECTURE Funcional OF PXOR IS
BEGIN
   C <=  A XOR B;
End Funcional;
```

**%%Arquivo Principal**

```
-- Library IEEE;
-- USE ieee.std_logic_1164.all;

ENTITY CIRCUITO IS
```

%% o cromossomo determina as entradas e as saídas do circuito

```
PORT( A, B   : IN bit;
     SA1 : OUT bit);

END CIRCUITO;
```

%% escreve no arquivo os componentes das operações lógicas.

```
ARCHITECTURE Estrutural OF CIRCUITO IS

   COMPONENT PNOT
       PORT( A : IN BIT;
```

```
            B : OUT BIT);
   END COMPONENT;

   COMPONENT FIO
       PORT( A : IN BIT;
             B : OUT BIT);
   END COMPONENT;

   COMPONENT PAND
       PORT( A,B  : IN BIT;
             C    : OUT BIT);
   END COMPONENT;

   COMPONENT POR
       PORT( A,B : IN BIT;
             C   : OUT BIT);
   END COMPONENT;

   COMPONENT PXOR
       PORT( A,B : IN BIT;
             C   : OUT BIT);
   END COMPONENT;
```

%% determina-se a quantidade de sinais pelo numero de genes (8) e entradas (2) do cromossomo.

%% entradas (S1 e S2) ..... .....

```
   SIGNAL S1, S2, S3, S4, S5, S6, S7, S8, S9, S10  : bit;
BEGIN
   I1:  FIO  PORT MAP (A,S1);
   I2:  FIO  PORT MAP (B,S2);
   I3:  PXOR  PORT MAP (S1,S2,S3);
   I4:  FIO  PORT MAP (S2,S4);
   I5:  FIO  PORT MAP (S3,S5);
   I6:  FIO  PORT MAP (S4,S6);
   I7:  PNOT PORT MAP (S5,S7);
   I8:  FIO  PORT MAP (S6,S8);
   I9:  FIO  PORT MAP (S7,S9);
   I10: FIO  PORT MAP (S8,S10);
   I11: FIO PORT MAP (S9,SA1);

END Estrutural;
```