BRUNO DE ABREU SILVA

METODOLOGIA PARA CRIAÇÃO DE UM SISTEMA DE HARDWARE EVOLUTIVO EM FPGA UTILIZANDO O PROCESSADOR NIOS II

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2008

BRUNO DE ABREU SILVA

METODOLOGIA PARA CRIAÇÃO DE UM SISTEMA DE HARDWARE EVOLUTIVO EM FPGA UTILIZANDO O PROCESSADOR NIOS II

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:

Hardware Evolutivo

Orientador:

Wilian Soares Lacerda

LAVRAS
MINAS GERAIS – BRASIL
2008

Ficha Catalográfica

Silva, Bruno de Abreu

Metodologia para Criação de um Sistema de Hardware Evolutivo em FPGA Utilizando o Processador NIOS II / Bruno de Abreu Silva. Lavras – Minas Gerais, 2008. 74p : il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1.Hardware Evolutivo. 2.FPGA. 3.Processador NIOS II. I. SILVA, B. A. II. Universidade Federal de Lavras. III. Metodologia para Criação de um Sistema de Hardware Evolutivo em FPGA Utilizando o Processador NIOS II.

BRUNO DE ABREU SILVA

METODOLOGIA PARA CRIAÇÃO DE UM SISTEMA DE HARDWARE EVOLUTIVO EM FPGA UTILIZANDO O PROCESSADOR NIOS II

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 21 de novembro de 2008

_____

Prof. André Vital Saúde

_____

Prof. Cláudio Fabiano Motta Toledo

_____

Prof. Wilian Soares Lacerda
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

"Não receie crescer devagar; só tenha medo de permanecer imóvel."
(Ditado chinês)

Agradeço a Deus, minha família, meus amigos e a todas as pessoas que de uma forma ou de outra contribuiram para a realização deste trabalho.
Agradecimento especial a FAPEMIG pelo apoio financeiro destinado ao Projeto de Pesquisa.

METODOLOGIA PARA CRIAÇÃO DE UM SISTEMA DE HARDWARE EVOLUTIVO EM FPGA UTILIZANDO O PROCESSADOR NIOS II

RESUMO

Esta pesquisa se enquadra na área de Computação Evolutiva aplicada no projeto e implementação de circuitos eletrônicos digitais. O objetivo principal é apresentar uma metodologia para criação de uma plataforma de *Hardware* Evolutivo em uma FPGA (*Field Programmable Gate Array*) utilizando Algoritmos Genéticos e o processador NIOS II. Neste trabalho foi desenvolvido um protótipo de um sistema de *Hardware* Evolutivo capaz de realizar a adaptação *online* de circuitos eletrônicos digitais combinacionais através da evolução dos circuitos. Com isso, foi possível desenvolver um sistema digital autônomo capaz de realizar a sua própria evolução de acordo com uma funcionalidade desejada.

Palavras-chave: Hardware Evolutivo, FPGA, Processador NIOS II.

METHODOLOGY TO CREATE AN EVOLVABLE HARDWARE SYSTEM AT FPGA USING THE NIOS II PROCESSOR

ABSTRACT

This research is included in the area of Evolvable Computer applied on design and implementation of digital electronic circuits. The main goal is show a methodology to create an Evolvable Hardware platform at a FPGA (Field Programmable Gate Array) using Genetic Algorithms and the NIOS II Processor. In this work, it was developed an Evolvable Hardware System Prototype able to realize the online adaptation of combinatorial circuits through the circuits evolution. With this, it was possible to develop an autonomous digital system able to realize its own evolution in agreement with a desired funcionality.

Keywords: *Evolvable Hardware, FPGA, NIOS II Processor.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Nesta seção será dada uma pequena contextualização do presente trabalho e suas motivações. Logo em seguida, também são descritos seus principais objetivos. E ao final serão descritos os capítulos do presente trabalho.

1.1 Contextualização

Nos últimos anos, abriu-se uma nova possibilidade na aplicação de conceitos biológicos à tecnologia: a construção de dispositivos capazes de evoluir, definidos como máquinas darwinianas ou, como é chamado comumente, *Hardware* Evolutivo ou *Evolvable Hardware* (EHW). A implementação do EHW envolve conceitos e técnicas de Computação Evolutiva, Redes Neurais Artificiais (RNA), vida artificial e eletrônica de circuitos reconfiguráveis, voltados para a construção de sistemas autônomos, auto-adaptáveis e tolerantes a falhas. O grande incremento dos recursos computacionais e também a disponibilização de uma nova geração de dispositivos lógico-programáveis mais robustos a alterações permitiram um avanço significativo do EHW nos últimos anos.

Cada vez mais dispositivos de *hardware* mais complexos tem exigido muito conhecimento dos projetistas e dessa forma o custo e o tempo de projeto tem aumentado bastante. A partir disso, há uma grande necessidade de automatizar o processo de projeto de dispositivos eletrônicos de forma a tornar o processo mais independente dos conhecimentos de quem projeta. Além de ser utilizado para projetar o próprio *hardware* e implementa-lo de forma automática, com EHW também é possível desenvolver sistemas tolerantes a falhas.

Atualmente o EHW já encontra aplicação na arquitetura de computadores, controladores para próteses médicas e impressoras fotográficas, entre outras.

Tradicionalmente, o EHW é muitas vezes implementado utilizando uma técnica da Computação Evolutiva conhecida como Algoritmos Genéticos para realizar a evolução do *hardware*.

1.2 Objetivos do Trabalho

O objetivo principal deste trabalho é apresentar uma metodologia para a criação de um sistema completamente autônomo de *Hardware* Evolutivo em FPGA (*Field Programmable*

Gate Array) utilizando Algoritmos Genéticos e o processador NIOS II (o NIOS II é um processador utilizado em sistemas desenvolvidos para FPGAs fabricadas pela *Altera Corporation*, será descrito em mais detalhes no Capítulo 6). Desenvolver um protótipo de um sistema de *Hardware* Evolutivo capaz de realizar a adaptação *online* de circuitos eletrônicos digitais combinacionais através da evolução dos circuitos. Com isso, deve ser possível desenvolver um sistema digital autônomo capaz de realizar a sua própria evolução de acordo com uma funcionalidade desejada. Completamente autônomo significa que o sistema será capaz de detectar mudanças no ambiente no qual estará submetido e automaticamente se adaptar através da reconfiguração da sua própria estrutura do *hardware*. Além desse objetivo, o presente trabalho visa atender também aos seguintes objetivos:

- Estudo das técnicas de projeto de circuitos digitais utilizando evolução de *hardware*. Considerando que esta técnica é atual, pouco ainda se sabe sobre esta nova metodologia;
- Implementação das técnicas de evolução de circuitos em *software* e em *hardware*;
- Realizar comparações com pelo menos dois tipos de Algoritmo Genético com o intuito de observar qual tipo é mais ideal.

1.3 Estrutura do Presente Trabalho

O Capitulo 1 chamado Introdução é apresentada uma pequena contextualização do presente trabalho mostrando o significado e a importância do EHW para o crescimento da Ciência da Computação, além de mostrar as motivações para se desenvolver pesquisas em EHW. Em seguida deixa claro quais são os objetivos que se deseja alcançar com este trabalho. E ao final, mostra qual é a estrutura do trabalho apresentando quais são os seus capítulos e quais assuntos são tratados nos mesmos.

No Capítulo 2 é mostrado como surgiram os Algoritmos Evolutivos, quais foram seus primeiros pesquisadores. Apresenta os fundamentos dos Algoritmos Evolutivos, e se aprofunda em um dos Algoritmos Evolutivos – os chamados Algoritmos Genéticos. Esse capítulo explica passo a passo o funcionamento de um *Algoritmo Genético Simples* com figuras e exemplos. Esse capítulo é de extrema importância uma vez que serve de base para que se possa entender o conceito de EHW.

No Capítulo 3 são apresentados conceitos básicos em sistemas digitais. Além de apresentar o que são portas lógicas e tabela-verdade, este capítulo também mostra técnicas tradicionais de modelagem e minimização de circuitos eletrônicos digitais a partir de uma tabela-verdade.

Já no Capítulo 4 pode-se encontrar o que são dispositivos de lógica programável, qual é a idéia base do funcionamento dos mais populares e como surgiram.

Tanto o objetivo do Capítulo 3 quanto do Capítulo 4 é apresentar conceitos que servem de base para introduzir o EHW uma vez que os dois pilares do EHW são os Algoritmos Evolutivos e a Eletrônica.

Finalizando os capítulos de Referencial teórico, o Capítulo 5 introduz o conceito de EHW, mostrando algumas definições comumente presentes nas literaturas além de apresentar como pode ser classificado um sistema de EHW. Este capítulo também cita os principais acontecimentos nessa área de pesquisa e os experimentos mais importantes, ou seja, o Estado da Arte.

O Capítulo 6 tem por objetivo mostrar qual é a metodologia utilizada para tornar possível o alcance dos objetivos listados na Introdução do presente trabalho. Apresenta as fases do desenvolvimento do trabalho e o que compõe cada fase. Este capítulo também tem por função esclarecer em que tipo de pesquisa este trabalho se enquadra no que diz respeito à natureza, aos objetivos e aos procedimentos.

O Capítulo 7 descreve quais os resultados alcançados ao seguir a metodologia descrita no Capítulo 6. Apresenta também uma breve discussão sobre os resultados obtidos e propostas de trabalhos futuros e sugestões de melhorias para o presente trabalho.

Por fim no Capítulo 8 estão presentes a conclusão deste trabalho.

2 ALGORITMOS EVOLUTIVOS

Por volta da década de 1940, cientistas começaram a se inspirar na natureza para tentar simular ou imitar a inteligência humana. Surgiu a partir daí o que veio a ser chamado de Inteligência Artificial (IA). IA procura desenvolver técnicas que viabilizem ao sistema computacional uma interação com o ambiente parcialmente conhecido e que se modifica ao longo do tempo. Geralmente, a IA se concentra em problemas que não respondem a soluções algorítmicas e dos quais não se tem todas as informações necessárias. Como os principais fundamentos da IA, pode-se citar:

- Representação do conhecimento;
- Busca dentro do espaço de soluções do problema.

A representação do conhecimento deve ser capaz de transformar as informações disponíveis do mundo real em dados computáveis, ou seja, a representação do conhecimento é um modelo computacional que representa todas as características consideradas relevantes para resolver o determinado problema.

As técnicas de busca sempre utilizam um processo inteligente ou adaptativo que trabalha com informações obtidas em estados anteriores para decidir a direção futura da busca.

Até o final da década de 1950, as pesquisas se desenvolveram mais no ramo cognitivo e na compreensão dos processos de raciocínio e aprendizado. Posteriormente, começou-se a procura por sistemas genéricos que fossem capazes de resolver problemas extremamente difíceis para se resolver computacionalmente.

Segundo Araújo (2004), devido à multiplicidade dos estudos na área da IA, está sendo construída uma hierarquia de teorias que caracterizam a IA de acordo com seu nível de abstração. São eles:

1. Nível baixo: Redes Neurais Artificiais e modelos emergentes (baseados em analogias biológicas). A abordagem nesse nível é totalmente empírica.
2. Nível médio: Abordagens que utilizam indução ou outras formas de raciocínio lógico.
3. Nível alto: Sistemas especialistas.

A Computação Evolutiva – ou Algoritmos Evolutivos – se enquadra na mesma categoria das Redes Neurais Artificiais. Os Algoritmos Evolutivos surgiram inspirados na teoria da evolução das espécies proposta por Darwin (1859). Cientistas começaram a desenvolver técnicas baseadas no princípio da evolução para resolver problemas computacionais complexos. Esse princípio diz que os animais e plantas que existem hoje são o resultado de milhões de anos de adaptação ao ambiente. Em um dado momento, diferentes organismos podem co-existir e competir por algum recurso num ecossistema. Os organismos mais capazes de obter os recursos e de procriar com sucesso são aqueles que terão o maior número de descendentes no futuro.

Técnicas de Computação Evolutiva podem ser usadas para buscar por soluções ótimas de um determinado problema. Em um algoritmo de busca, um número de possíveis soluções para o problema são avaliadas e o objetivo é encontrar a melhor solução possível. Para uma busca em um espaço com poucas soluções, pode-se avaliar todas as soluções e então compara-las para saber qual é a ótima sem demandar muito tempo computacional. Mas em problemas cujo espaço de busca é consideravelmente grande (tipicamente, problemas da classe *NP-Completo*), esta *busca exaustiva* torna-se inviável. Para resolver esse tipo de problema, são utilizados algoritmos aproximativos ou heurísticas. As técnicas de Computação Evolutiva, segundo Linden (2006), são classificadas como heurísticas pois são algoritmos polinomiais que não tem garantia nenhuma sobre a qualidade da solução encontrada, mas que na prática encontram a solução ótima ou uma solução bem próxima da ótima.

Um dos diferenciais dos Algoritmos Evolutivos em relação às heurísticas tradicionais é que eles são baseados em população. Através da adaptação de sucessivas gerações de uma população de indivíduos, um Algoritmo Evolutivo desempenha um eficiente direcionamento na busca.

Em um Algoritmo Evolutivo típico, um *modelo de representação* é escolhido pelo pesquisador para definir a forma de representação de todas as soluções possíveis para o problema e assim definir o espaço de busca para o algoritmo. É gerado (geralmente de forma aleatória) um número de soluções individuais para comporem a população inicial. Os passos descritos a seguir são repetidos de forma iterativa até que algum *critério de parada* pré-definido seja satisfeito. Cada indivíduo da população é avaliado pela *função de avaliação (fitness)* que é uma função específica para cada problema a ser resolvido. Com base nos resultados da

avaliação, escolhe-se os indivíduos que serão os *pais*. Através dos *operadores de reprodução*, novos indivíduos são gerados a partir dos pais e formam os *descendentes*. Por fim, sobreviventes são selecionados dentre os pais e descendentes para formar a nova população da próxima *geração*. Muitos métodos consideram que o número de indivíduos a cada geração é fixo.

Outro conceito comum aos Algoritmos Evolutivos é o que se chama de *método de seleção.* É o método de seleção que irá determinar quantos e quais pais serão selecionados para formar os descendentes, quantos descendentes serão gerados e quais indivíduos sobreviverão na próxima geração.

De acordo com Hussain (1998) e Bäck & Schwefel (1996), os Algoritmos Evolutivos mais populares são: Estratégias Evolutivas, Programação Evolutiva, Programação Genética e Algoritmos Genéticos. Na prática, todas essas técnicas se misturam, pois existe uma variedade de técnicas sendo pesquisadas como representações mais complexas, operadores novos como foi o caso da *recombinação elitista* para os Algoritmos Genéticos, em que os pais competem com os filhos para formarem a próxima geração. As vantagens de cada técnica ainda estão sendo analisadas e exploradas.

2.1 Estratégias Evolutivas

A representação usada é um vetor de valores reais de tamanho fixo. Cada posição do vetor é uma característica do indivíduo. O principal operador de reprodução é a *mutação Gaussiana*, em que um valor aleatório de uma distribuição Gaussiana é adicionado para cada elemento do vetor para criar um novo descendente. Existe também outro operador, o *intermediate recombination*. Nesse operador os vetores dos pais são mediados juntos, elemento por elemento para formar um novo indivíduo.

A formação da nova geração é feita da seguinte forma: N pais são aleatoriamente selecionados, são gerados vários descendentes (mais do que N), e então os N melhores indivíduos dentre os descendentes e os pais são os selecionados.

2.2 Programação Evolutiva

Comumente, também utiliza uma representação com vetor de valores reais de tamanho fixo. Mas somente utiliza a mutação como operador de reprodução. Para a representação com

vetor de valores reais, esse método se assemelha muito com as Estratégias Evolutivas sem recombinação.

Para realizar a seleção, um método bastante comum é selecionar todos os indivíduos da população para serem os *N* pais, realizar a mutação nos *N* pais gerando *N* descendentes. E então através do *fitness* escolher *N* indivíduos dos *2N* existentes para formar a próxima geração.

2.3 Algoritmos Genéticos

Além do que já foi dito no início da seção 2 , de acordo com Soares (1997) os Algoritmos Genéticos são um método Estocástico, ou seja, baseado em regras de probabilidade para resolver problemas. Um Algoritmo Genético se aplica bem na resolução de problemas em que a busca pela solução exata demanda muito tempo computacional.

Os Algoritmos Genéticos têm ganhado popularidade por serem bem eficientes. Como a busca pela solução usa regras de probabilidade, a chance de se encontrar uma solução global é grande. Também não é necessário ter conhecimento matemático aprofundado do problema, o que representa uma grande vantagem, por exemplo, com relação aos métodos Determinísticos.

Outras vantagens podem ser citadas como:

- Tolerância a ruídos e dados incompletos;
- Pode funcionar com parâmetros contínuos e/ou discretos;
- São fáceis de serem paralelizados;
- Sua implementação em computadores é relativamente simples;
- Uma vez implementado pode ser transferido de um problema para outro já que o mecanismo de evolução é independente da modelagem do problema;

- São flexíveis para trabalhar com objetivos conflitantes.

Inicialmente nos Algoritmos Genéticos, as representações costumavam ser uma cadeia de *bits* de tamanho fixo, embora não seja a mais eficiente. No caso dessa representação clássica, cada posição da cadeia representa uma característica particular de um indivíduo (de forma análoga aos genes de um organismo biológico).

O primeiro Algoritmo Genético foi introduzido por Holland e popularizado por Goldberg (1989), é chamado de *Algoritmo Genético Simples* (SGA, do inglês *Simple Genetic Algorithm*) e seu fluxograma típico pode ser visto na Figura 2.1.

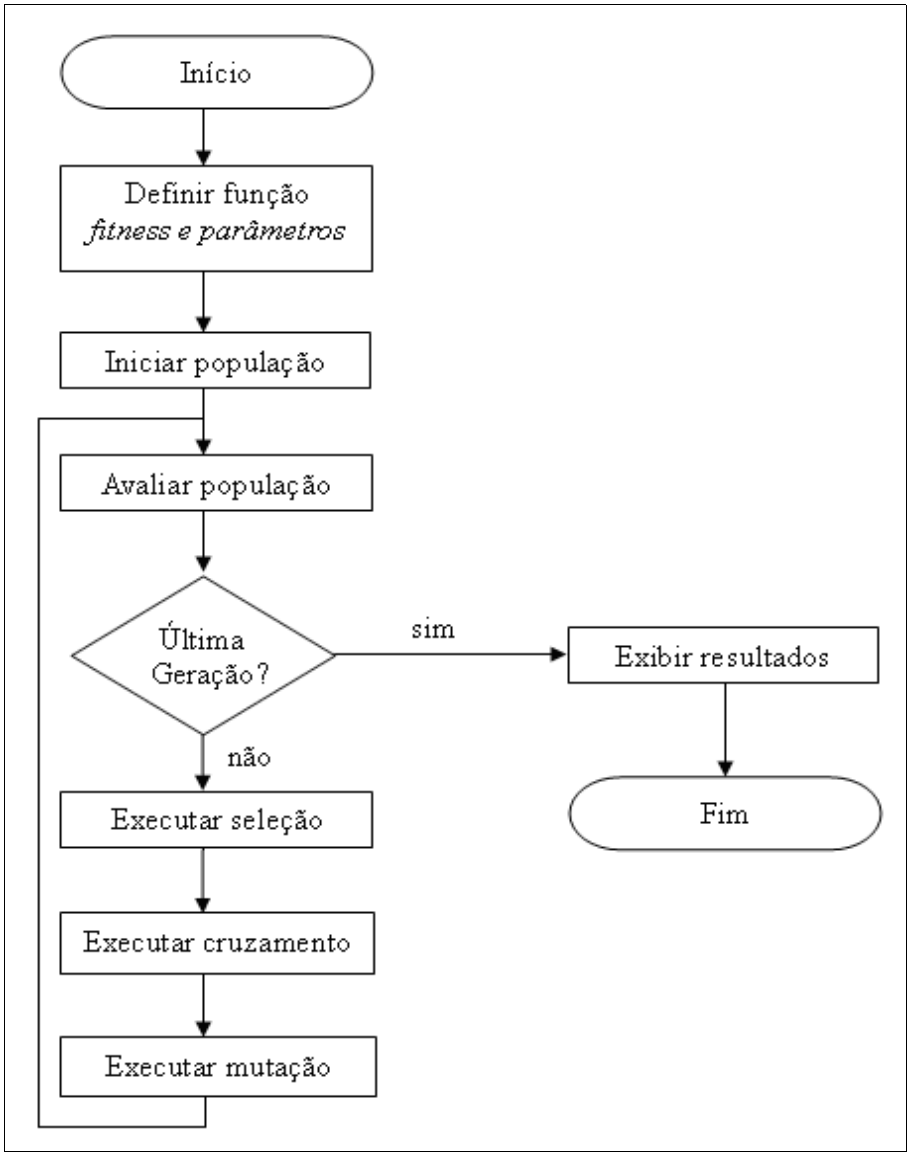


Figura 2.1: Fluxograma de um Algoritmo Genético

FONTE: Dados do Trabalho

A função desempenho ou *fitness* deve ser uma função que dado um indivíduo como entrada retorne um valor associado a esse indivíduo de modo que os indivíduos mais aptos possuam maiores valores de *fitness* para problemas de maximização ou valores menores para problemas de minimização.

Em um Algoritmo Genético, a seleção, o cruzamento e a mutação são os operadores de reprodução. Na realidade, qualquer aproximação computacional de fenômenos que ocorrem na natureza (no contexto evolutivo) são chamados de operadores de reprodução ou operadores genéticos.

Para conseguir representar uma possível solução adequadamente deve-se codifica-la. Cada solução é representada por uma concatenação de variáveis, os cromossomos. E cada cromossomo pode ser também codificado. Na codificação binária, o cromossomo pode ser um vetor de binários no qual cada posição é um gene, o valor desse gene (0 ou 1) é o alelo e a posição do gene no vetor é chamado de lócus de acordo com Soares (1997).

Além da codificação binária, existem outras codificações. Por exemplo, em Skliarova & Ferrari (2002), um problema clássico da Computação, o Problema do Caixeiro Viajante, foi resolvido utilizando Algoritmos Genéticos. Nesse caso, cada indivíduo poderia ser uma seqüência de números inteiros, onde cada número representa uma cidade visitada e, a ordem na qual as cidades aparecem nessa seqüência corresponde a ordem de visitação.

Voltando à codificação binária, considere, por exemplo, um indivíduo que represente um número inteiro x = 22.

Pode-se codificar esse indivíduo, por exemplo, com um cromossomo que consiste em um vetor de binários com 5 posições (Veja a figura 2.2).

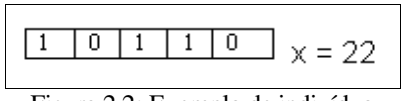


Figura 2.2: Exemplo de indivíduo

FONTE: Dados do Trabalho

Deve-se definir também as probabilidades dos operadores. Por exemplo, se for definida a probabilidade de cruzamento 60% e a de mutação 10%, significa que, se a população for formada por 100 indivíduos, a cada geração aproximadamente 60 cruzamentos acontecerão, e 10 indivíduos sofrerão a mutação (a probabilidade de mutação pode ser aplicada por gene,

nesse caso cada gene teria 10% de chance de sofrer uma mutação). Ao fim de cada iteração do *loop* considera-se que se passou uma geração.

Os critérios de convergência mais utilizados na literatura são:

- O número máximo de gerações;
- O alcance de um determinado erro entre a solução encontrada e a solução desejada;
- Verifica-se se não houve melhora do melhor indivíduo da geração atual em relação ao melhor indivíduo da geração anterior por *n* gerações.

Os indivíduos são avaliados para que a seleção possa escolher os melhores. Um método de seleção muito comum é o método da Roleta. Nesse método, os indivíduos com melhores valores para a função desempenho têm maiores probabilidades de serem selecionados. Imagine que existem quatro indivíduos A, B, C e D com valores de probabilidades de serem selecionados iguais a 50%, 10%, 10% e 30% respectivamente (Figura 2.3). A idéia deste método é girar a roleta e selecionar o indivíduo relativo à área onde a roleta parou de girar. Sorteiam-se vários indivíduos até que se tenha o número de indivíduos que se deve ter na população.

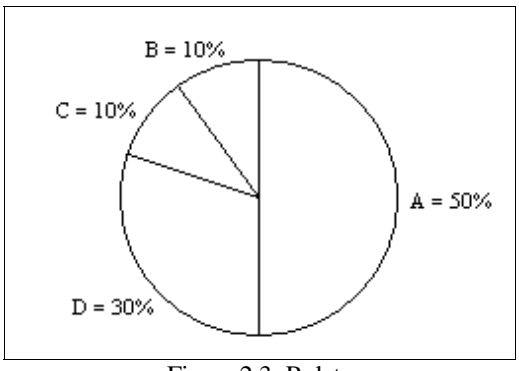


Figura 2.3: Roleta

FONTE: Dados do Trabalho

Uma possível execução do método da Roleta para os indivíduos A, B, C e D em uma população de quatro indivíduos, poderia ser:

- Primeiro sorteio: Indivíduo A;
- Segundo sorteio: Indivíduo D;
- Terceiro sorteio: Indivíduo A;

- Quarto sorteio: Indivíduo B.

Outro método de seleção muito comum é o Torneio. Nesse método dois indivíduos são escolhidos aleatoriamente da população e o indivíduo com melhor *fitness* é selecionado. É um método muito usado, pois é extremamente simples implementa-lo.

Após a seleção ser executada, entra em ação o operador de cruzamento. Para que fique mais claro o funcionamento deste operador, será apresentado um exemplo. Considere, como na figura 2.4, dois indivíduos codificados com um vetor de binários conforme já descrito:

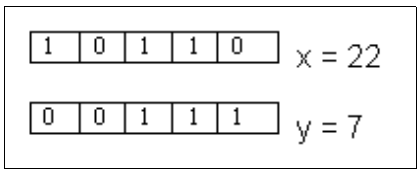


Figura 2.4: Exemplo de dois indivíduos

FONTE: Dados do Trabalho

Nesse caso, considere que a troca de material genético *(crossover*) ocorrerá a partir do terceiro gene – os pontos que delimitam a região onde ocorrerá a troca são conhecidos como pontos de corte, nesse caso há somente um ponto de corte, a troca ocorre a partir dele.

Como se pode observar na Figura 2.5, após o cruzamento entre os indivíduos x e y, obtém-se x' = 23 e y' = 6.

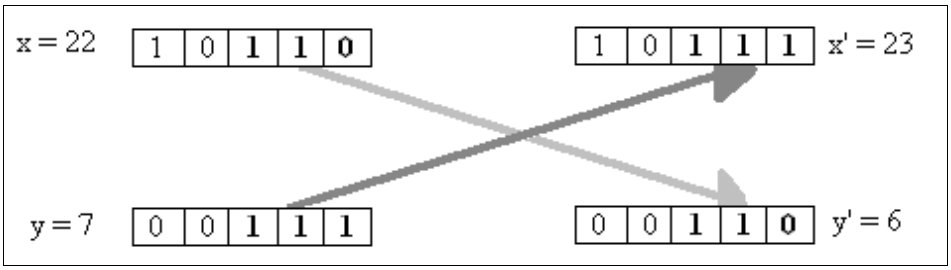


Figura 2.5: Exemplo de cruzamento

FONTE: Dados do Trabalho

Existem outros tipos de cruzamento, como por exemplo o cruzamento uniforme que realiza uma varredura em cada posição do vetor de *bits* (no caso da codificação binária) dos pais escolhendo aleatoriamente qual pai doará o *bit* para o filho de modo que ao final tem-se como resultado um indivíduo que é uma mistura uniforme dos genes dos pais.

Antes do final de cada geração, temos a chamada do operador de mutação. O operador de mutação altera algum valor da codificação do indivíduo, como por exemplo, trocar o valor de um determinado *bit* de 0 para 1. O objetivo da mutação é perturbar uma dada solução para tentar manter uma certa divergência entre os membros da população e explorar áreas ainda inexploradas do espaço de busca. Ao longo das gerações, a mutação é um operador de equilíbrio extremamente importante, pois com o cruzamento e a seleção a tendência é que a população se torne cada vez mais homogênea. Isso pode fazer com que o algoritmo convirja para mínimos/máximos locais e fique preso, não conseguindo alcançar o ótimo global.

2.4 Programação Genética

Tradicionalmente, na Programação Genética, a representação usada é uma árvore de funções e valores de tamanho variável. Programação Genética é muito parecida com os Algoritmos Genéticos. A principal diferença é que na Programação Genética os operadores manipulam árvores. A Programação Genética é muito utilizada para criar programas de computador de forma automática utilizando como representação árvores de derivações de alguma gramática pré-definida.

2.5 Conclusão

A Computação Evolutiva tem crescido consideravelmente nos últimos anos, sendo aplicada para resolver os mais diversos problemas. Devido a isso, muita pesquisa vem sendo realizada nessa área. Segundo Linden (2006), atualmente várias outras técnicas foram mescladas com os Algoritmos Genéticos para poder melhorar sua eficiência. Obviamente, deve-se procurar técnicas que melhor resolvem um determinado problema que está sendo atacado, pois como já foi dito, os Algoritmos Genéticos são classificados como uma técnica empírica. Os Algoritmos Genéticos são facilmente hibridizados e paralelizados, ampliando assim a gama de possibilidades existentes para pesquisa nessa área.

Outros operadores tem se consolidado como fundamentais para se alcançar bons resultados. Um deles é o elitismo (ou *recombinação elitista*), que consiste em preservar na próxima geração o melhor indivíduo da geração atual sem nenhuma alteração. Há também o operador de epidemia, não tão comum quanto o elitismo, mas é bastante interessante. Seu princípio é matar vários indivíduos em uma dada população e gerar novos indivíduos aleatórios. A combinação de elitismo e epidemia podem ser bastante interessantes.

3 CIRCUITOS ELETRÔNICOS DIGITAIS

Circuitos eletrônicos digitais são circuitos que funcionam baseados em lógica binária, ou seja, a informação é codificada e guardada com apenas zeros (0) e uns (1). O nível lógico zero é representado por uma tensão baixa (tipicamente 0 *volts*) e o nível lógico um é representado por uma tensão diferente de 0 *volts* (tipicamente 5 *volts*). Existem dois tipos de circuitos eletrônicos digitais: os circuitos seqüenciais e os circuitos combinacionais.

Os circuitos combinacionais são aqueles cujos valores das saídas dependem única e exclusivamente do estado dos valores das entradas no atual momento. Os circuitos combinacionais não armazenam informação, eles apenas tomam decisões.

Os circuitos seqüenciais são aqueles cujos valores das saídas dependem tanto dos estados atuais dos valores das entradas quanto de valores anteriores das entradas. Esse tipo de circuito armazena informações, ou seja, possui capacidade de memorização e tomam suas decisões baseadas em momentos anteriores e no momento atual.

Atualmente, grande parte da tecnologia disponível é baseada em circuitos eletrônicos digitais, ou sistemas digitais, como por exemplo: computadores, redes de telecomunicações e sistemas embarcados. Devido à complexidade assumida por esses sistemas, novas técnicas de projeto têm sido pesquisadas com o intuito de diminuir a dependência da habilidade do projetista no ambiente de projeto através da elevação do nível de automatização durante o projeto.

3.1 Tabela-verdade

Tabela-verdade é uma forma de representar como um circuito eletrônico digital irá se comportar sem se preocupar com a sua estrutura. Consiste em uma tabela com uma coluna para cada entrada e uma coluna para cada saída. Para cada combinação dos *bits* da entrada é mostrado quais os *bits* desejados para as saídas. Veja a Figura 3.1 que mostra a tabela-verdade de um somador binário completo (sem *Carry out*). Nessa tabela, têm-se três entradas (*Carry In*, *Bit 1* e *Bit 2*) e uma saída (*Soma*). Para as 8 combinações possíveis dos três *bits* de entrada é mostrada na coluna da saída qual é o *bit* desejado.

Carry In	Bit 1	Bit 2	Soma
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Figura 3.1: Tabela-verdade

FONTE: Dados do Trabalho

Não necessariamente a tabela precisa conter todas as combinações possíveis, basta que contenha somente as combinações relevantes para o problema em questão. Uma combinação ou uma entrada que não importa na tabela-verdade comumente é chamada de *don't care* e é representada por "-" ou por "x", ou no caso de uma combinação inteira não importar ou se sua saída desejada for desconhecida, ela é simplesmente omitida da tabela.

3.2 Portas Lógicas

Para desenvolver um circuito eletrônico digital basicamente utiliza-se alguma combinação dos três operadores lógicos fundamentais: AND ($Entrada1 \wedge Entrada2$), OR ($Entrada1 \vee Entrada2$) e NOT ($\neg Entrada$), que são conhecidos como portas lógicas (Figura 3.2). Qualquer sistema combinacional pode ser implementado utilizando um primeiro nível de portas NOT, um segundo nível de portas AND e um terceiro nível de portas OR, ou seja, por uma expressão *Booleana* em forma de soma de produtos, também chamada de expressão *Booleana* na forma normal disjuntiva. De acordo com Ercegovac et al. (2000) essa expressão é composta por:

- Literais: Variáveis complementadas e não-complementadas. A porta NOT obtém na sua saída o complemento do literal recebido como entrada;
- Produtos: Estes são implementados utilizando-se a porta AND;
- Soma: O nível da soma é implementado por portas OR.

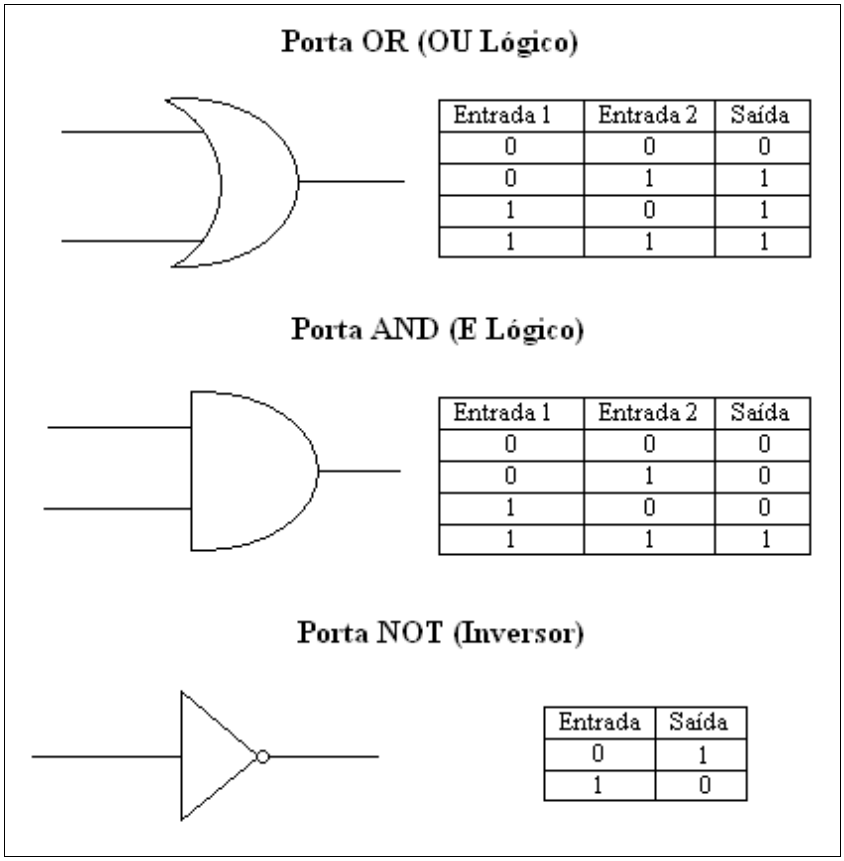


Figura 3.2: Portas lógicas e suas tabelas-verdade

FONTE: Dados do Trabalho

Veja um exemplo de expressão *Booleana* na forma normal disjuntiva:

$$(B \wedge C) \vee (A \wedge B) \vee (A \wedge C) \vee (A \wedge B \wedge C)$$

Para transformar uma expressão *Booleana* na forma normal disjuntiva em um circuito eletrônico digital, basta substituir cada literal por uma entrada e fazer cada ligação na porta lógica correspondente como mostra a Figura 3.3.

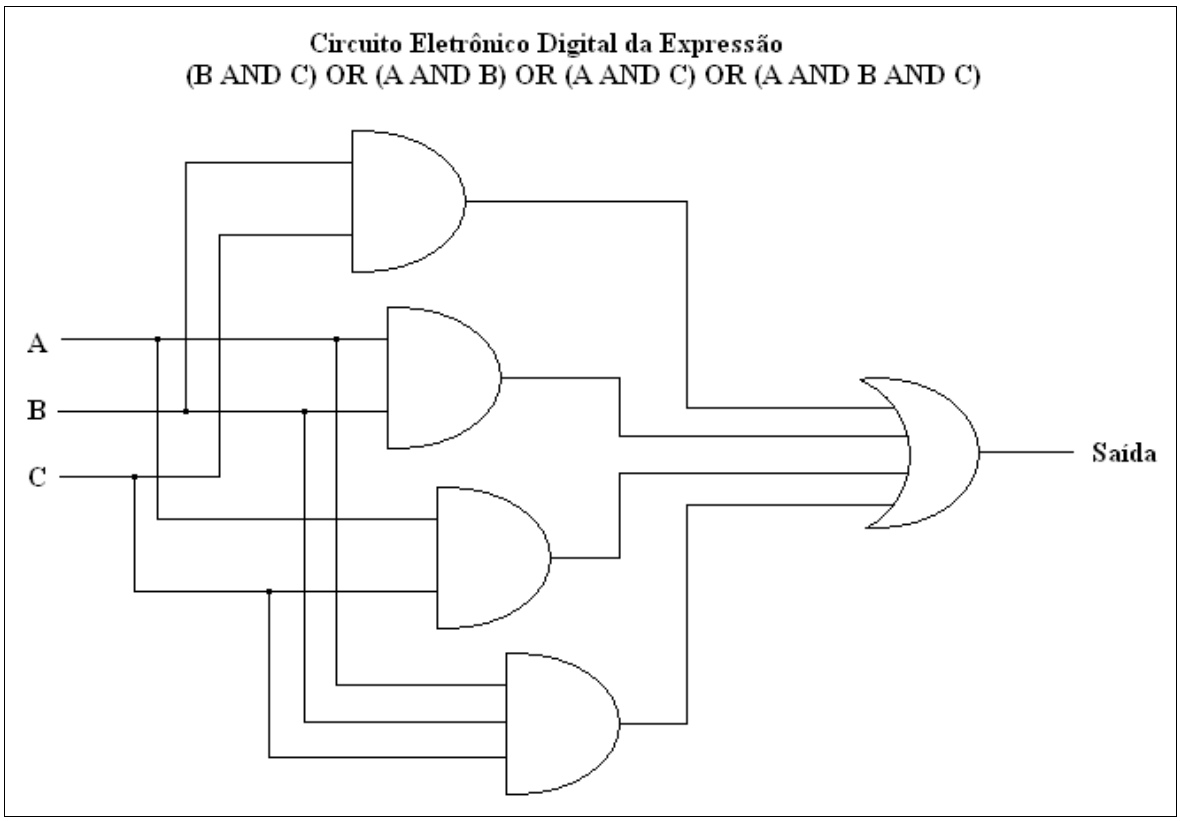


Figura 3.3: Circuito eletrônico digital

FONTE: Dados do Trabalho

3.3 Modelagem e Minimização de Circuitos a Partir da Tabela-verdade

O primeiro passo para modelagem e desenvolvimento de um circuito eletrônico digital é construir sua tabela-verdade de acordo com o comportamento desejado. Tendo em mãos a tabela-verdade, existem técnicas para extrair da tabela uma expressão *Booleana* na forma normal disjuntiva que representa o circuito que se comporta exatamente como orienta a tabela-verdade. As técnicas geralmente geram um circuito que obedece completamente a tabela-verdade, porém nem todas encontram um circuito que possui o menor número de portas e conexões possíveis, ou seja, o circuito mínimo que obedece a tabela-verdade. Outras técnicas sempre encontram o circuito mínimo, porém demandam muito tempo computacional para tabelas-verdade razoavelmente grandes. De acordo com Silva et al. (2004), encontrar uma função *Booleana* que obedeça uma tabela-verdade e que seja mínima é um problema classificado como *NP-Completo*. Em outras palavras, não existe algoritmo exato conhecido capaz de

resolver esse problema em tempo polinomial. O tempo de execução dos algoritmos exatos para resolver esse problema cresce exponencialmente em função do tamanho da tabela-verdade, tornando-se inviável já para tabelas relativamente pequenas. Geralmente algoritmos aproximativos ou heurísticas são utilizados para resolver os problemas classificados como *NP-Completo*.

Explicar como obter uma soma de produtos (expressão *Booleana* na forma normal disjuntiva) mínima requer as seguintes definições, segundo Ercegovac et al. (2000):

- **Implicante:** Um termo de produto *p* é um implicante de uma função se, para qualquer atribuição para a qual *p* tenha o valor 1, a função também tem o valor 1. Qualquer termo do produto que aparece em uma soma de produtos que represente a função é um implicante dessa função. Isto acontece porque a expressão tem valor 1 sempre que o termo do produto tenha valor 1;

- **Implicante primo:** Dado o conjunto de valores onde a saída do implicante gera 1 de acordo com a função *Booleana*, o implicante primo é aquele que possui um conjunto que não é subconjunto de nenhum outro conjunto de qualquer implicante da função;

- **Implicante primo essencial:** Um implicante primo é essencial se existir uma atribuição *a* tal que $p(a) = 1$ e $d(a) = 0$ para qualquer outro implicante primo *d*.

Todos os implicantes primos essenciais são incluídos em uma soma de produtos mínima. Mas a soma de produtos não tem necessariamente somente eles, muitas vezes devem possuir implicantes primos não-essenciais para que a soma de produtos esteja de acordo com a tabela-verdade. Deve-se tentar escolher aqueles implicantes primos não-essenciais de menor custo.

A seguir serão descritas algumas das técnicas mais conhecidas para extrair uma expressão *Booleana* de uma tabela-verdade.

3.3.1 A Técnica de Modelagem Mais Simples

A técnica mais simples para extrair uma expressão *Booleana* de uma tabela-verdade é pegar todas as linhas cuja saída é 1 e ligá-las através do operador lógico OR e cada entrada dessas linhas são ligadas através do operador lógico AND, as entradas com *bit* zero são negadas. Está

técnica encontra um circuito que obedece a tabela-verdade, porém para tabelas grandes pode gerar expressões muito grandes, o que muitas vezes inviabiliza a implementação do circuito. Ou seja, esse método provavelmente não encontrará o menor circuito possível, pois não utiliza nenhuma técnica de minimização, o que na prática faz com que essa técnica não seja tão utilizada. Para a tabela-verdade do somador binário completo (Figura 3.1) obtém-se a seguinte expressão *Booleana*:

$$Soma = (\neg CI \wedge \neg B1 \wedge B2) \vee (\neg CI \wedge B1 \wedge \neg B2) \vee (CI \wedge \neg B1 \wedge \neg B2) \vee (CI \wedge B1 \wedge B2)$$

Esta técnica apresenta um limitante superior para o espaço de busca do circuito mínimo, pois o circuito mínimo será sempre menor ou igual ao circuito obtido por esse método.

3.3.2 Mapa de Karnaugh

Conforme Tocci & Widmer (2003), o mapa de Karnaugh é um diagrama que além de extrair uma expressão *Booleana* que obedece a tabela-verdade, a expressão também é a mínima. Portanto, é um método que sempre encontra a solução ótima para o problema de minimização, ou seja, é um método exato. Porém é um método limitado. É um excelente método para até cinco ou seis variáveis de entrada. Quando se aumenta esse número, torna-se muito complexo aplicar tal método. É altamente recomendável para problemas com mais de seis variáveis utilizar soluções algorítmicas computacionais.

Para o caso do somador binário completo, o mapa de Karnaugh encontra a mesma expressão *Booleana* que o método mostrado anteriormente, isso porque não existem agrupamentos possíveis na sua tabela-verdade (consulte Tocci & Widmer (2003) para ver detalhes sobre como aplicar o mapa de Karnaugh).

3.3.3 Método Tabular de Quine McCluskey

O método tabular de Quine McCluskey é recomendável para se obter uma soma de produtos mínima em problemas com mais de seis variáveis e para casos em que se deseja utilizar um computador para fazer a minimização. O método de Quine McCluskey é dividido em duas etapas:

1. Determinação dos implicantes primos da função;
2. Escolha de um conjunto de implicantes primos que cubra a função dada e tenha o custo mínimo.

A segunda etapa do Quine McCluskey é justamente a parte do problema de minimização de expressões *Booleanas* que já foi dito anteriormente ser *NP-Completo*. Trata-se de um problema de Otimização Combinatória. Por ser um método exato, o método de Quine McCluskey torna-se rapidamente inviável de ser executado conforme aumenta-se o número de variáveis de entrada do problema.

Não é objetivo deste trabalho entrar em detalhes sobre o funcionamento desse método tabular, porém em Ercegovac et al. (2000) há um exemplo de como o método é executado.

3.3.4 ESPRESSO

O ESPRESSO é um minimizador de equações *Booleanas* extremamente eficiente e rápido. É um dos métodos mais usados atualmente. Cabe ressaltar que o ESPRESSO é um método heurístico, ou seja, não garante a qualidade da solução. Porém sua saída costuma ser mínima em cardinalidade e irredundante. Consulte Lacerda (2006) para maiores informações sobre este método.

3.4 Conclusão

Atualmente, a teoria base dos sistemas digitais está bastante consolidada. Porém, algumas limitações começam a aparecer na prática com o aumento da complexidade dos circuitos a serem projetados e o aparecimento de dispositivos lógico-programáveis. Essas limitações tem gerado uma necessidade de realização de pesquisas de soluções alternativas para projeto de sistemas digitais.

4 DISPOSITIVOS DE LÓGICA PROGRAMÁVEL

Atualmente a maioria dos *hardwares* dos dispositivos eletrônicos possuem estrutura fixa. São projetados e implementados para realizar uma função específica e após serem implementados permanecem executando a mesma função pelo resto de sua vida útil. Os computadores também possuem *hardware* fixo. Os processadores desses computadores são projetados para serem processadores de propósito geral, o que pode fazer com que esses processadores possuam desempenho pior ao executar uma tarefa específica do que um processador projetado especificamente para realizar tal tarefa.

Porém, esforços recentes tem gerado uma nova geração de dispositivos de *hardware* com o objetivo de melhorar o poder de processamento das máquinas. Não se trata de desenvolver um *hardware* fixo exclusivo para realizar cada tarefa, mas trata-se de dispositivos de lógica programável (*PLD – Programmable Logic Devices*). Esses dispositivos possuem um *hardware* configurável que pode ser reprogramado para alguma aplicação que esteja executando. Ou seja, um dispositivo de lógica programável permite que o *hardware* seja mais flexível e ainda chegue próximo do desempenho de um dispositivo projetado especificamente para uma única tarefa.

Os dispositivos de lógica programável são muito utilizados nos dias de hoje. As principais vantagens dos *Programmable Logic Devices* são:

- Facilidade de desenvolvimento de protótipos;
- Baixo custo de desenvolvimento;
- Produção rápida;
- Facilidade de modificação.

O primeiro *chip* programado pelo usuário que poderia implementar circuitos lógicos foi a *Programmable Read Only Memory* (PROM). Funções lógicas, entretanto utilizam poucos termos de produto geralmente, e a PROM possui um decodificador completo para seus endereços, o que a torna ineficiente para este propósito.

4.1 PLAs e PALs

O primeiro dispositivo desenvolvido com o objetivo específico de implementar funções lógicas foi o *Programmable Logic Array* (PLA), Figura 4.1. PLA possui dois níveis de portas lógicas, AND e OR que podem ser configuradas pelo usuário. Com sua estrutura é possível representar funções lógicas na forma de soma de produtos. A PLA foi introduzida no início da década de 1970 pela *Philips*. Nessa época era difícil de se fabricar PLAs e elas ainda introduziam um atraso significativo aos circuitos. Para contornar este problema, a *Monolithic Memories* (MMI) desenvolveu a *Programmable Array Logic* (PAL), Figura 4.2. As PALs são um pouco mais simples do que as PLAs, pois são configuráveis apenas em um nível. O nível do AND alimenta o nível do OR, que é fixo.

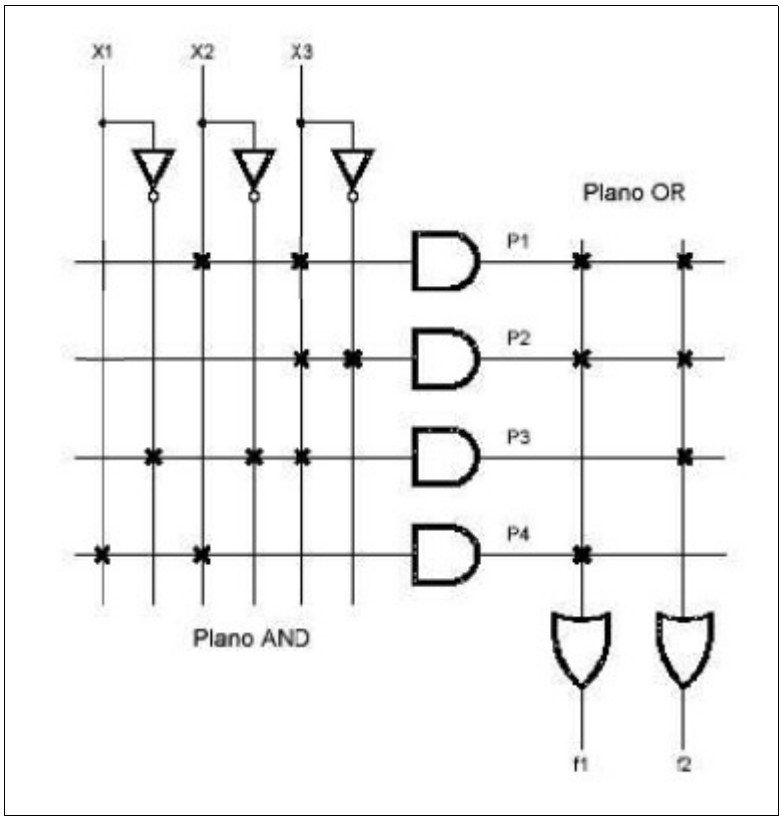


Figura 4.1: Diagrama simplificado de uma PLA

FONTE: Dados do Trabalho

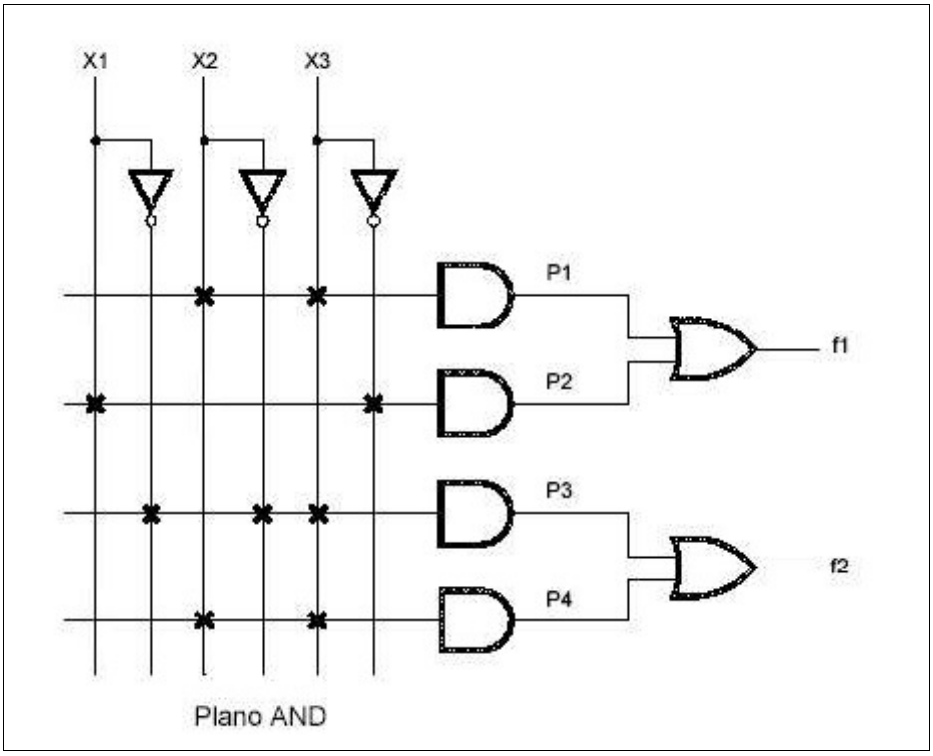


Figura 4.2: Diagrama simplificado de uma PAL

FONTE: Dados do Trabalho

Como se pode ver na Figura 4.2, cada ponto onde se tem o cruzamento de uma porta AND com uma entrada (negada ou não) indica se essa entrada faz parte do AND ou não. Tipicamente nesses pontos se têm fusíveis. Para indicar a ausência de uma entrada simplesmente queima-se o seu fusível correspondente.

4.2 CPLDs e FPGAs

Posteriormente, a *Altera Corporation* desenvolveu os CPLDs (*Complex Programmable Logic Devices*), que são vários dispositivos simples como a PAL ou PLA em um único *chip*. E após vários estudos e avanços, surgiu a FPGA (*Field Programmable Gate Array*) que consiste em uma matriz de elementos de circuitos independentes (blocos lógicos) e interconexões, porém é o usuário final quem configura a FPGA através de programação e determina qual função específica a FPGA irá realizar. Com as FPGAs tornou-se viável implementar circuitos mais complexos em PLDs deixando o projeto de circuitos eletrônicos digitais ainda mais fácil e flexível. As FPGAs podem ser reconfiguradas várias vezes graças à tecnologias de chaveamento que podem ser implementadas por EPROMs (*Electrical Programmable Read Only*

Memory), EEPROMs (*Electrical Eraseble Programmable Read Only Memory*), SRAMs (*Static Read Only Memory*), etc.

Devido a complexidade dos blocos lógicos, ferramentas de *software* são necessárias para programar estes dispositivos.

Os principais fabricantes de FPGA são:

- Xilinx;
- Actel;
- Altera;
- Quick Logic.

A Figura 4.3 apresenta a estrutura padrão de uma FPGA. Cada caixa retangular mostrada na figura representa um bloco lógico configurável. As linhas entre os blocos representam os recursos de interconexão. Os blocos lógicos de entrada e saída são especificados como roteamento das entradas e saídas de sinais.

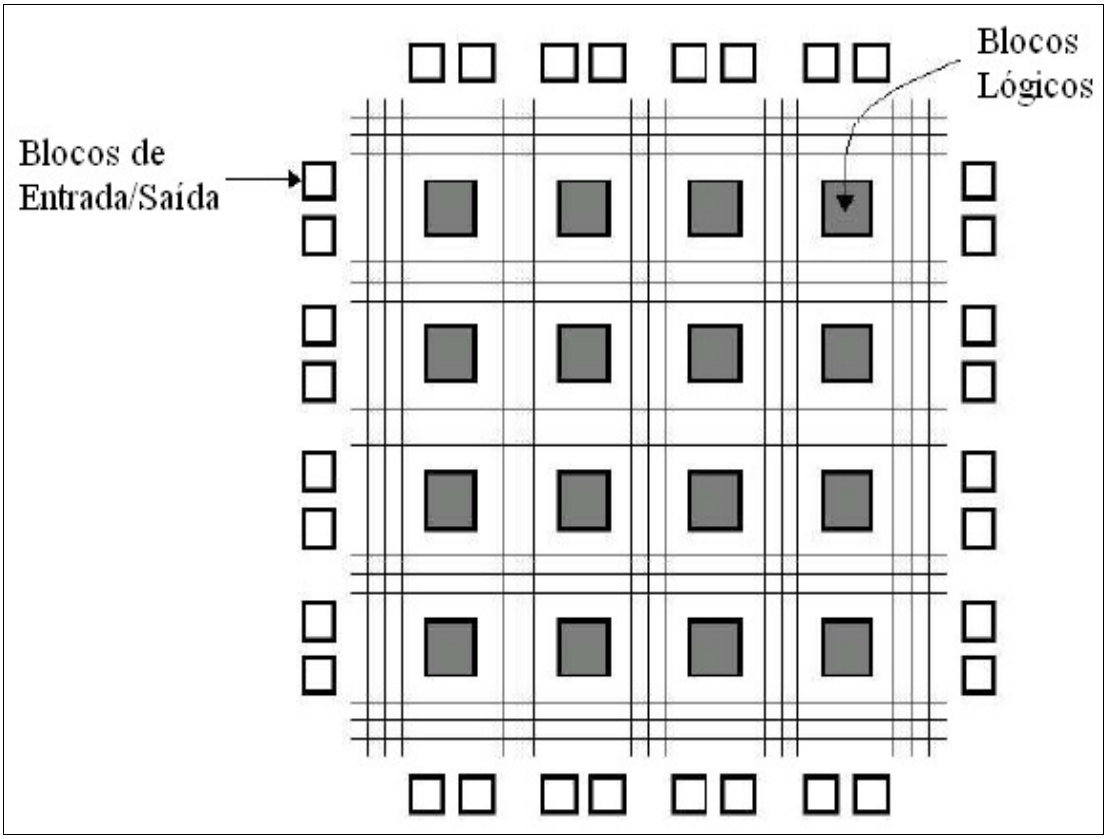


Figura 4.3: Estrutura padrão de uma FPGA

FONTE: Dados do Trabalho

4.3 Outros PLDs

A tecnologia eletrônica atual disponibiliza também os seguintes dispositivos de lógica programável, entre outros:

- PLS – *Programmable Logic Sequence;*
- GAL – *Gate Array Logic.*

Uma GAL possui as mesmas propriedades lógicas de uma PAL, porém ela pode ser apagada e reprogramada. A matriz de fusíveis é usada para determinar as interconexões entre entradas do dispositivo e as células lógicas como também para especificar as entradas dos termos de produto das células lógicas.

Muitos PLDs são programados através de linguagens de descrição de *hardware* (HDL, do inglês *Hardware Description Language*). Essas linguagens descrevem como é a arquitetura do *hardware*, seu funcionamento, seu comportamento. As linguagens de descrição de *hardware* mais comuns são Verilog e VHDL, pois são uma linguagem de mais alto nível, fator determinante para a eficiência do desenvolvimento de sistemas mais complexos.

4.4 Conclusão

O desenvolvimento dos PLDs foi de fundamental importância para impulsionar as pesquisas na área de sistemas digitais e também sistemas embarcados. Por facilitar e acelerar consideravelmente o processo de projeto, PLDs têm sido amplamente usados para o desenvolvimento de protótipos. Também tornou mais fácil realizar testes com técnicas da Inteligência Artificial, viabilizando a criação de sistemas autônomos e inteligentes mais robustos.

Com os PLDs, o *hardware* se tornou mais poderoso, pois agora pode ser reaproveitado para diferentes funções. Atualmente, existem técnicas que permitem que o *hardware* inclusive se adapte de forma autônoma e *online* às mudanças de função necessárias impostas pelo ambiente no qual está inserido, como por exemplo o *Hardware* Evolutivo que será apresentado a seguir.

5 *HARDWARE* EVOLUTIVO

Diferentemente do *hardware* convencional, o Hardware Evolutivo possui a capacidade de adaptar-se a mudanças nas tarefas requeridas ou ao ambiente, através de sua habilidade para reconfigurar sua própria estrutura de *hardware* dinamicamente e automaticamente.

Em De Garis (1991), foi introduzido o conceito de *Hardware* Evolutivo (*Evolvable Hardware* – EHW). Para De Garis, dispositivos capazes de evoluir são essenciais no desenvolvimento de sistemas nervosos artificiais para que robôs possam desenvolver algum tipo de aprendizado, a partir de interações com estímulos ambientais. É uma área de grande interesse tanto para o campo da Inteligência Artificial (IA) quanto para a vida artificial.

EHW é o uso da Computação Evolutiva para síntese e implementação de circuitos eletrônicos capazes de resolver problemas do mundo real. A adaptação tem a forma da modificação direta da estrutura do *hardware* de acordo com o ambiente. Como vantagens se tem a adaptação em tempo real e sistemas flexíveis e tolerantes a falhas, pois o sistema poderá mudar sua própria estrutura caso o ambiente mude ou em caso de erro de *hardware*.

A definição de EHW também pode ser separada em duas definições:

1. Aplicações de técnicas evolutivas para síntese de circuitos.
2. *Hardware* que é capaz de realizar uma adaptação *online* através da reconfiguração da sua própria arquitetura de forma autônoma e dinâmica.

Na primeira definição existem duas fases distintas na sua implementação, uma fase é a da evolução do *design* e a outra é a fase de execução, que usualmente não requer adaptação *online* (embora seja possível). Para ativar a adaptação *online* o EHW deve adaptar sua arquitetura enquanto opera no ambiente real.

EHW é fundamentalmente diferente de implementação de Algoritmos Evolutivos em *hardware* onde a arquitetura não muda e é usado para implementar funções de seleção, recombinação e mutação. A principal motivação para a implementação de Algoritmos Evolutivos em *hardware* é para aumentar a velocidade de execução de suas funções.

O termo *Hardware* Evolutivo é usado geralmente em aplicações envolvendo evolução em *chips* reconfiguráveis (PLDs).

Para que o conceito de EHW seja útil, é necessário satisfazer ao menos duas condições:

- Os circuitos que evoluem devem ser, simultaneamente, funcionais e muito complexos para a compreensão humana, caso contrário poderiam ser projetados através das técnicas tradicionais;

- Os circuitos devem evoluir mais rápido que a evolução simulada em *software*, caso contrário seria mais fácil e prático realizar apenas simulações.

5.1 Taxonomias em *Hardware* Evolutivo

Segundo Albert (1997), o EHW poderia ser dividido em duas categorias: *fine-grained e function-level*. No primeiro caso, usa-se um Algoritmo Evolutivo para selecionar componentes ou valores dos componentes num circuito analógico ou criar circuitos digitais a partir de blocos de construção com portas. Já no segundo caso, o EHW geralmente envolve a seleção de topologias e montagem de circuitos a partir de blocos de construção funcionais de alto nível. Obviamente, o espaço de soluções no *function-level* é geralmente menor que o do *fine-grained*, pois os blocos de função usados são conhecidos pelo pesquisador como necessários e suficientes para resolver o problema.

De acordo com o projeto eletrônico, o EHW pode ser classificado em três categorias:

- Digital;
- Analógico;
- Híbrido (Analógico e Digital).

Outra taxonomia de EHW, que se refere ao uso de simuladores de circuitos ou *chips* reconfiguráveis como plataformas para o processo de busca, pode ser descrita como:

- Evolução extrínseca ou indireta: evolução realizada por simulação e então a melhor solução é implementada em *hardware* reconfigurável, isto é, o *hardware* é configurado apenas uma vez;

- Evolução intrínseca ou direta: evolução diretamente em *hardware* e em tempo real, isto é, todo indivíduo será usado para reconfigurar o *hardware*. O EHW será reconfigurado o mesmo número de vezes quanto o tamanho da população em cada geração.

5.2 Estado da Arte

Em meados de 1980, pesquisadores perceberam que técnicas evolutivas poderiam ser usadas para otimização e roteamento de componentes e placas de circuitos. O primeiro trabalho que introduziu a idéia de Algoritmos Evolutivos como ferramenta para realizar o projeto de estrutura de circuitos digitais foi Louis e Rawlins como consta em Louis & Rawlins (1991), quando Algoritmos Evolutivos foram utilizados para interligar portas digitais que poderiam resolver um problema específico, tal como função de paridade. No entanto, no início de 1990, pesquisadores começaram a aplicar tais técnicas para o desenvolvimento da arquitetura e projeto de circuitos utilizando a tecnologia dos PLDs, que continham uma coleção de módulos lógicos digitais (blocos funcionais) cujas interconexões poderiam ser programadas pelo usuário. Uma representação foi usada para codificar as interconexões dos blocos que permitiram a evolução de circuitos via Algoritmos Evolutivos. A idéia era que cada indivíduo na população representasse uma configuração específica no PLD. Para avaliar o indivíduo deveria-se configura-lo no PLD e verificar seu funcionamento no *hardware*. Porém PLDs eram muito limitadas pelo ciclo lento de programação e por poderem ser reconfiguradas um número limitado de vezes, restringindo assim o poder dos primeiros experimentos nessa área.

Também no início de 1990, devido ao surgimento das FPGAs, circuitos mais complexos puderam ser evoluídos. Mas devido ao fato das arquiteturas das FPGAs serem proprietárias, o avanço do EHW continuou no nível funcional. Entretanto, um acontecimento importante em meados de 1990 permitiu que o *fine-grained* tivesse avanços significativos: A empresa Xilinx, fabricante de FPGAs, introduziu a Unidade de Processamento Reconfigurável. Isso viabilizou a capacidade de interconectar células arbitrariamente com muitas topologias flexíveis. Nessa mesma época, pesquisadores focaram seus estudos em circuitos *fine-grained* analógicos. Eles evoluíram circuitos analógicos em simulação usando *softwares* simuladores, como por exemplo o SPICE. Porém, simuladores de circuitos são lentos e a avaliação destes circuitos não pode ser feita em tempo real.

Em 1994, Higushi et al. (1994) propuseram uma arquitetura de EHW que utiliza uma GAL16V8 e descrevem uma série de experimentos feitos por eles os quais incluíam um multiplexador, um contador binário e uma máquina de estados finita, Higushi et al. (1996) evoluíram circuitos digitais para reconhecimento de padrões e confirmaram através do Problema do OU-Exclusivo que EHW pode reconhecer padrões de funções não-linearmente separáveis. O

maior problema encontrado pelos pesquisadores nessa época era que o tamanho dos circuitos evoluídos era muito pequeno e por isso a dificuldade de transformar o EHW em algo aplicável nas industrias. Em 1996, Thompson (1996) evoluiu intrinsecamente um circuito em uma FPGA.

O experimento pioneiro de Thompson em 1996 impulsionou o EHW e é um marco histórico na área. Ele evoluiu um circuito (discriminador de freqüência) usando uma FPGA, resultando em um circuito mais econômico do que o projetado por humanos. A tarefa era evoluir um circuito em uma FPGA, para discriminar entre onda quadrada de 1kHz e outra de 10kHz presente na entrada. A saída deveria ir para +5 volts quando uma freqüência estivesse presente, e 0 volts quando fosse a outra.

Uma FPGA foi utilizada, onde transistores utilizados como chaves controlam como cada componente se comporta, e como os componentes são conectados entre si. O estado das chaves são controlados pelo conteúdo da memória RAM com *bits* distribuídos através do *chip*. Esta memória pode ser rapidamente e repetidamente escrita, causando um vasto número de possibilidades de circuitos eletrônicos a serem fisicamente implementados. Cada projeto candidato era um conjunto de *bits* de configuração para a matriz de células. Para testar seu *fitness*, cada projeto foi configurado em um FPGA real e rajadas de sinais de 1kHz e 10 kHz foram aplicadas a uma entrada (Figura 5.1). Uma pontuação do *fitness* foi automaticamente associada conforme a saída se aproximava do comportamento de discriminação desejado. Durante a evolução do circuito, a cada nova geração o melhor circuito (indivíduo) se aproximava mais da função desejada (Figura 5.2).

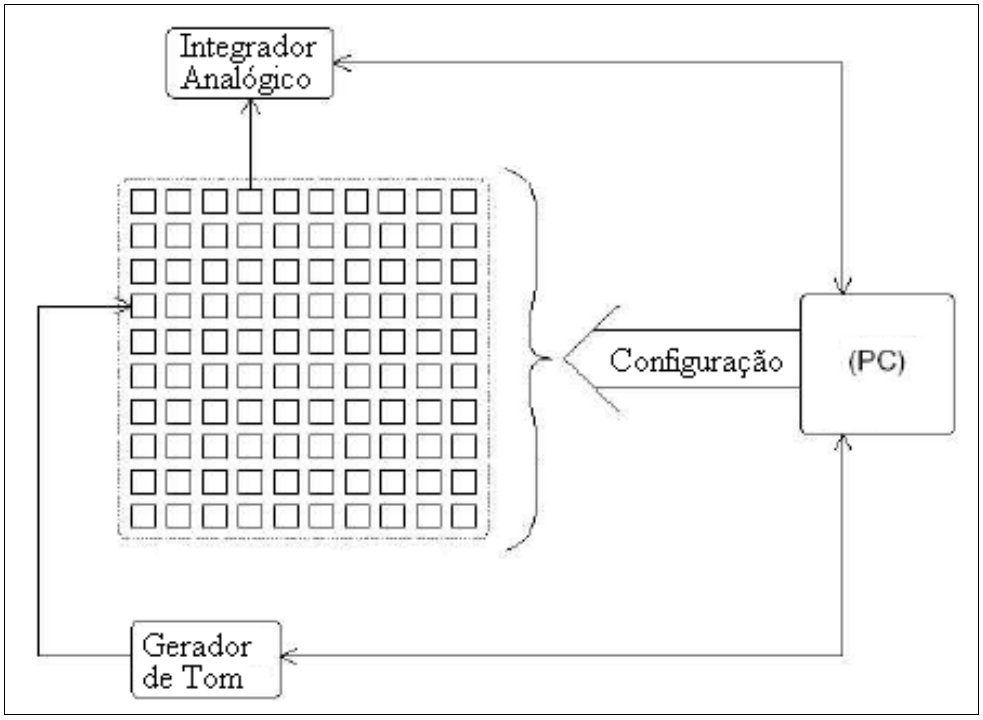


Figura 5.1: Arranjo para evolução do circuito discriminador em FPGA

FONTE: Adaptado de Thompson (1996)

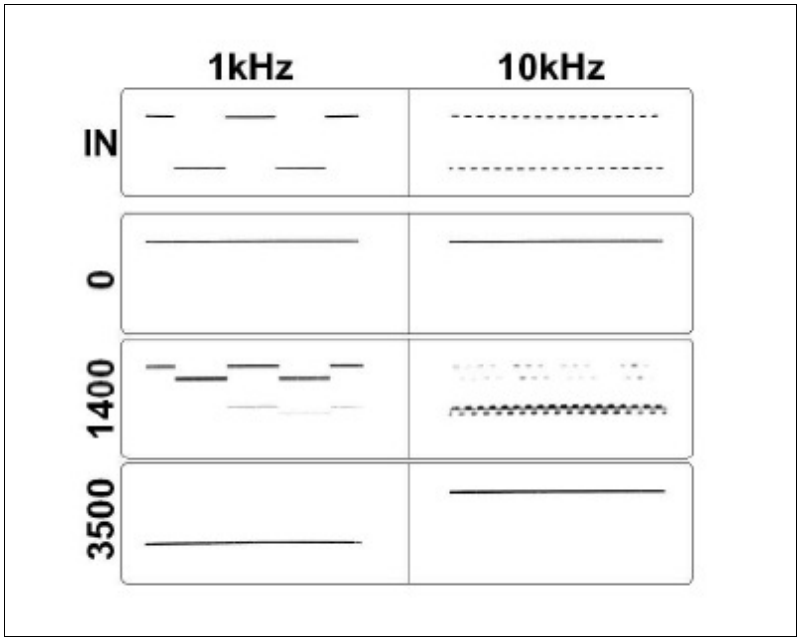


Figura 5.2: Formas de onda de saída do discriminador em cada geração

FONTE: Thompson (1996)

Tentou-se realizar simulações do circuito usando o programa SPICE, e a simulação nunca reproduziu o comportamento do circuito da FPGA.

Se o circuito fosse configurado em outra FPGA nominalmente idêntica, então seu desempenho era degradado. Se a temperatura aumenta ou diminui, então o circuito ainda funciona, mas as freqüências entre as quais ela discrimina são aumentadas ou diminuídas.

Este experimento deixou claro que muita pesquisa fundamental ainda tinha que ser concluída.

Mas os pesquisadores começaram a perceber que EHW poderia substituir as Redes Neurais Artificiais em aplicações industriais, pois a técnica da Computação Evolutiva elimina a necessidade de selecionar uma arquitetura de RNA e treinamento e foca em apenas dois requisitos: a habilidade para representar uma solução num formato genético e a habilidade para avaliar o *fitness* relativo de uma dada solução. Além disso as implementações são geralmente mais compactas do que com Redes Neurais Artificiais. Pesquisadores também argumentam que o reconhecimento de padrões por EHW é muito mais rápido que em Redes Neurais Artificiais porque a função classificadora está implementada diretamente em lógica combinacional.

A partir do final da década de 90, começaram a surgir algumas aplicações reais para EHW. Em Higushi et al. (1999), são mostrados *chips* de EHW e várias aplicações que estavam sendo desenvolvidas como parte do projeto *MITI's Real-World Computing Project* como uma rede neural em *chip* EHW capaz de reconfiguração autônoma, *chip* analógico para celulares, um *chip* EHW para compressão de dados em uma impressora fotográfica entre outros.

A área de pesquisa em EHW começava a se consolidar, seus desafios e suas promessas instigavam vários pesquisadores pelo mundo pois estava começando a se tornar prática para algumas aplicações do mundo real, especialmente em problemas fora do domínio das metodologias tradicionais. Em Yao & Higushi (1999), são apresentadas algumas questões que ainda estavam abertas até o momento. Trabalhos em EHW estavam concentrados sobre o projeto evolutivo de circuitos digitais embora o objetivo era desenvolver *hardware* com adaptação *online*. Porém, poucos estudos relatavam sobre EHW que adaptava sua arquitetura e função enquanto opera num ambiente real.

Um trabalho muito interessante também foi feito já no início dos anos 2000 por Levy et al. (2000), onde foi descrito um sistema de EHW capaz de reconhecer dígitos decimais escritos

a mão. O sistema ainda era capaz de se adaptar a diferentes usuários. O protótipo foi desenvolvido em uma FPGA utilizando a linguagem de descrição de *hardware* VHDL. VHDL começou a ser muito utilizado em sistemas de EHW pois possibilitava a implementação no nível funcional via Programação Genética, como foi o caso de Araújo (2004) onde os indivíduos representavam programas em VHDL. A desvantagem é que para avaliar cada indivíduo precisa-se compila-lo e executa-lo, isso pode ser muito lento para problemas complexos.

Muitos pesquisadores passaram a adotar as FPGAs como tecnologia alvo para os EHW digitais por elas proverem uma plataforma reconfigurável e serem comercialmente disponíveis. As características mais desejadas para as FPGAs no EHW digital são:

- Flexibilidade e velocidade – Flexibilidade com relação ao número de reconfigurações possíveis e velocidade com relação ao número de avaliações e operações que ela é capaz de fazer por unidade de tempo;
- Reconfiguração parcial;
- Auto-reconfiguração – Habilidade de uma parte do *chip* para reconfigurar outra parte do mesmo *chip*.

5.3 Conclusão e Desafios Atuais do *Hardware* Evolutivo

As técnicas do EHW permitem explorar projetos alternativos não criados por humanos usando técnicas convencionais, além disso podem trabalhar com requisitos especiais e variados graus de restrição, bastando incorporar à função de avaliação e ao cromossomo do Algoritmo Evolutivo. E também não necessitam de conhecimento matemático específico do domínio do problema. Mas o EHW é muito pequeno com poucos componentes em comparação com os circuitos desenvolvidos pelos métodos convencionais. E mesmo para esses pequenos EHW, tem-se experimentado um alto custo computacional para realizar a evolução dos circuitos.

Circuitos avaliados em ambientes físicos reais no EHW intrínseco podem causar severos danos ao sistema ou ao ambiente, e em contrapartida a evolução simulada (EHW extrínseco) somente manipula a sintaxe e não a semântica do circuito codificado gerando avaliações imprecisas. Uma função de avaliação correta muitas vezes representa um custo computacional inviável para os sistemas de tempo real e aplicações práticas.

Outro problema encontrado pelos pesquisadores é que os EHW são algumas vezes difíceis de serem entendidos e analisados pois podem não ter uma clara decomposição funcional. Isso dificulta seriamente a manutenibilidade do sistema. Sistemas feitos por humanos podem ser entendidos pela decomposição funcional por causa da abordagem "dividir para conquistar" universalmente adotada para projetos complexos. Embora um sistema evoluído poder ter funções particulares localizadas e subsistemas identificáveis, isso não ocorre sempre. Teorias de sistemas dinâmicos oferecem um suporte matemático no qual sistemas podem ser caracterizados sem a decomposição funcional.

Pesquisadores, como Salami & Hendtlass (2005) e Stomeo (2006) tem dedicado esforços para desenvolver técnicas que melhorem a escalabilidade dos EHW e que diminua seu custo computacional, principalmente na avaliação dos circuitos.

6 METODOLOGIA

Este capítulo apresenta, inicialmente, o tipo de pesquisa em que se enquadra o presente trabalho no que diz respeita à natureza, aos objetivos e aos procedimentos. Em seguida, são apresentados os materiais e métodos que foram utilizados durante a execução do trabalho.

6.1 Tipo de Pesquisa

Seguindo o modelo apresentado por Jung (2004), com relação ao tipo da pesquisa, esta pode ser classificada: quanto à sua natureza como tecnológica, já que objetiva apresentar um processo ou uma metodologia para criação de um sistema de EHW; quanto aos objetivos como exploratória, já que tem por finalidade descobrir novas práticas e realizar estudos exploratórios na área de EHW; quanto aos procedimentos como experimental, já que utilizará protótipos, simulação e modelagem; e por fim classifica-se a princípio como pesquisa em laboratório, já que é possível controlar as variáveis que podem intervir no experimento.

Com relação ao tempo de aplicação do estudo, este estudo se caracteriza como transversal pois o estudo foi feito em um determinado instante do tempo. A pesquisa bibliográfica foi feita com base em livros, artigos de anais de congresso e periódicos. Foram utilizados também manuais para implementar o sistema que será explicado ao longo deste capítulo.

6.2 Kit de Desenvolvimento Utilizado

Para implementar o *Hardware* Evolutivo foi utilizado o kit de desenvolvimento da *Altera Corporation* conhecido como *Nios II Development Kit – Stratix II Edition*[1] que possui a seguinte FPGA: *Stratix II* – EP2S60F672C3N (Veja o kit na Figura 6.1).

A plataforma de desenvolvimento *Stratix II Edition* possui as seguintes características, além de outras que podem ser consultadas em Altera Corporation (2005):

- Mais de 13.500 módulos lógicos e 1,3 milhões de *bits* de memória *on-chip;*
- 16 *Mbytes* de memória *flash*;
- 2 *Mbytes* de SRAM síncrona;
- 32 *Mbytes* de DDR SDRAM;

1 Este kit foi comprado com financiamento da FAPEMIG especificamente para a realização do presente trabalho.

- Conector para depuração para *software* e *hardware*;
- Porta serial RS232;
- 4 botões para serem conectados por pinos de E/S (entrada/saída) da FPGA;
- Dois *displays* de 7 segmentos;
- 8 LEDs;
- Conector JTAG para que o kit possa se comunicar com um PC;
- Oscilador gerador de *clock* de 50 MHz;
- Dispositivo *on-board* de *Ethernet* MAC/PHY.

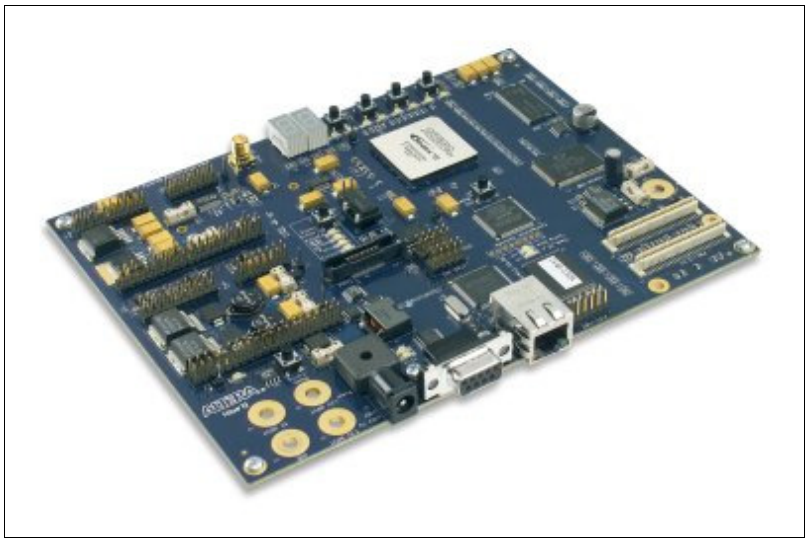


Figura 6.1: Kit de desenvolvimento Stratix II – EP2S60F672C3N

FONTE: Altera Corporation (2005)

6.3 Processador NIOS II

O NIOS II é um processador RISC de propósito geral que pode ser colocado em qualquer sistema de *hardware* desenvolvido para executar em FPGAs da *Altera Corporation* através do programa *Quartus II* integrado com o programa *SOPC Builder*[2]. Esse processador contem entre outras características que podem ser consultadas em Altera Corporation (2007a):

- Conjunto de instruções, dados e espaço de endereçamento de 32 *bits*;

2 *Quartus II* e *SOPC Builder* são programas fornecidos pela *Altera Corporation* para auxiliar no desenvolvimento de sistemas para suas FPGAs

- 32 registradores de propósito geral;
- 32 fontes de interrupção externa;
- Instruções simples de multiplicação e divisão em 32 *bits*;
- Instruções dedicadas para processamento de produtos de multiplicação de 64 *bits* e 128 *bits*;
- Acesso a uma variedade de periféricos *off-chip, on-chip* e interfaces para memória;
- Módulo para depuração através da integração com um ambiente de desenvolvimento;
- Desempenho em torno de 150 DMIPS (*Dhrystone Millions of Instruction per Second*).

O NIOS II é um processador cujo núcleo pode ser configurado por *software*. Características podem ser somadas ou subtraídas dele. Ou seja, o núcleo da CPU é construído através de projeto de *software*. Sua finalidade é ser implementado em alguma família de FPGAs. É o usuário quem configura o processador e seus periféricos de acordo com suas necessidades.

O processador NIOS II possui várias características que permitem aumentar o desempenho de algum sistema específico como:

- Instruções de multiplicação aceleradas por *hardware*;
- Pode conter memória *on-chip* com cache de instruções e de dados;
- O usuário pode construir instruções específicas para o processador;
- O NIOS II pode ser construído com quantos periféricos, interfaces, CPUs e memória forem necessários, dependendo apenas da quantidade de elementos lógicos presentes dentro da FPGA alvo.

Entre outras vantagens do NIOS II, ele também ocupa poucos blocos lógicos de uma FPGA de alta densidade. Geralmente consome algo em torno de 5%. Dessa forma, todos os blocos restantes podem ser usados para implementação de outras funções.

A *Altera Corporation* disponibiliza a maioria dos periféricos geralmente utilizados em microcontroladores, tais como:

- *Timers*;
- Interfaces de comunicação serial;
- Portas gerais de E/S;
- Controladores de SDRAM e SSRAM;
- Periféricos criados e personalizados pelo projetista.

A arquitetura do NIOS II define as seguintes unidades funcionais visíveis ao usuário, dentre outras:

- Controlador de exceção;
- Controlador de interrupção;
- Módulo de depuração JTAG.

O módulo de depuração JTAG disponibiliza emulação *on-chip* para controlar o processador remotamente através de um *host*. Ferramentas de *software* de depuração no *host* comunicam com o módulo JTAG e disponibilizam facilidades como *download* de programas para memória, configuração de *breakpoints* e *watchpoints*, análise de registradores e memória, coleta de dados em tempo de execução como arquivos ou dados fornecidos por teclado através do *host*.

O processador NIOS II suporta as seguintes famílias de FPGAs da *Altera Corporation*:

- *Stratix*;
- *Stratix II*;
- *Cyclone*;
- *Cyclone II*.

O ambiente de desenvolvimento de um sistema de *hardware/software* que utiliza o processador NIOS II pode ser resumido na Figura 6.2. Em resumo, o projeto do processador NIOS II é feito através do programa *SOPC Builder*, o processador é então integrado no projeto de *hardware* que é feito no programa *Quartus II*. O projeto do *hardware* é então compilado e logo em seguida pode ser programado em uma FPGA. O desenvolvimento do *software* do sistema é feito no programa *NIOS II IDE* que também é fornecido pela *Altera Corporation* e

uma vez pronto pode ser simulado pelo programa *ModelSim*[3], mas também pode ser executado já na FPGA que está configurada com o *hardware* do sistema.

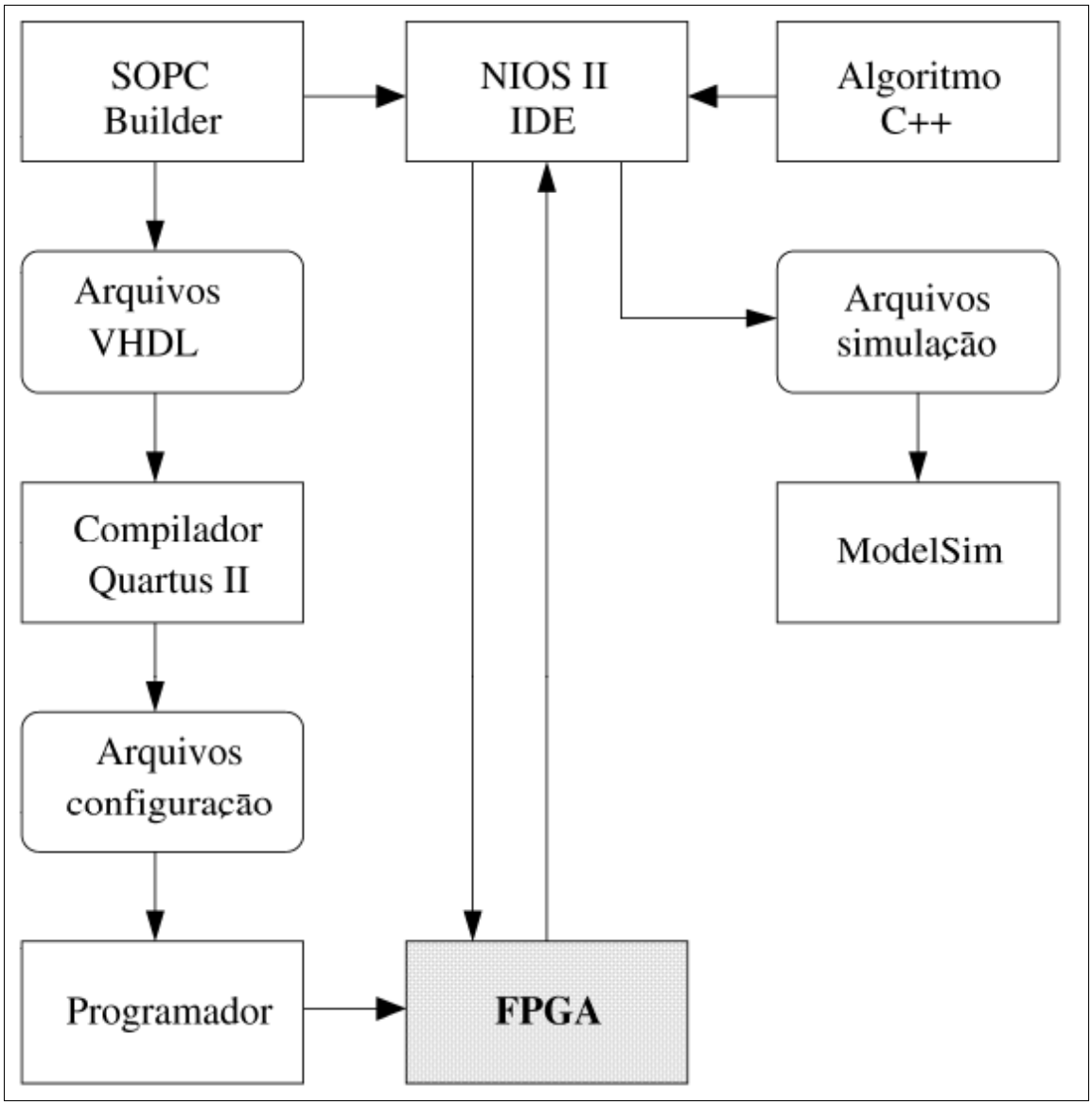


Figura 6.2: Ambiente de desenvolvimento de um sistema com NIOS II

FONTE: Lacerda (2006)

6.4 Visão Geral do Sistema

O sistema de EHW projetado é uma sistema que deve ser capaz de perceber uma necessidade de criação de um circuito eletrônico, deve possuir um módulo funcional capaz de gerar

3 *ModelSim* é marca registrada da *Mentor Graphics Corporation*

um circuito adequado e otimizado de acordo com a necessidade identificada, deve possuir um dispositivo de *hardware* que possa ser reconfigurado com o circuito encontrado. Por fim, o sistema deve ser capaz também de se adaptar a mudanças nas necessidades executando novamente a otimização do circuito e a reconfiguração do *hardware* a cada mudança de necessidade. Outra característica desejável é que o *hardware* reconfigurado continue funcionando enquanto o processo de evolução de outros circuitos ocorre.

Para atender aos requisitos desejados, as seguintes decisões de projeto foram tomadas:

- A entrada do sistema é um arquivo com uma tabela-verdade que descreve as características de um circuito digital combinacional arbitrário que representa a necessidade atual que o sistema deverá ser capaz de atender;
- O módulo otimizador de circuitos receberá a tabela-verdade e evoluirá, através de um Algoritmo Evolutivo (no caso, Algoritmo Genético) o melhor circuito possível;
- Após a evolução, o módulo envia somente o melhor circuito encontrado para ser configurado no dispositivo de *hardware*;
- O dispositivo de *hardware* escolhido para ser reconfigurado foi a PAL. A PAL é um dispositivo capaz de armazenar qualquer expressão *Booleana* na forma de soma de produtos dentro dos seus limites de tamanho. A PAL é poderosa pois é capaz de resolver qualquer problema que possa ser modelado como um circuito digital combinacional, inclusive reconhecimento de padrões binários como consta em Lacerda (2006);
- A PAL deve ter pinos de entrada e pino de saída reservados para o uso da mesma após sua configuração.

Assim o EHW implementado será classificado como digital, *function-level* (evolui circuitos lógicos combinacionais), indireto ou extrínseco (somente o melhor indivíduo será configurado no *hardware* após uma execução do Algoritmo Evolutivo) e com evolução *online* pois o sistema continua funcionando enquanto a evolução ocorre (viável para continuar executando enquanto opera em ambiente físico real).

O ciclo do funcionamento do sistema pode ser visto no fluxograma da Figura 6.3.

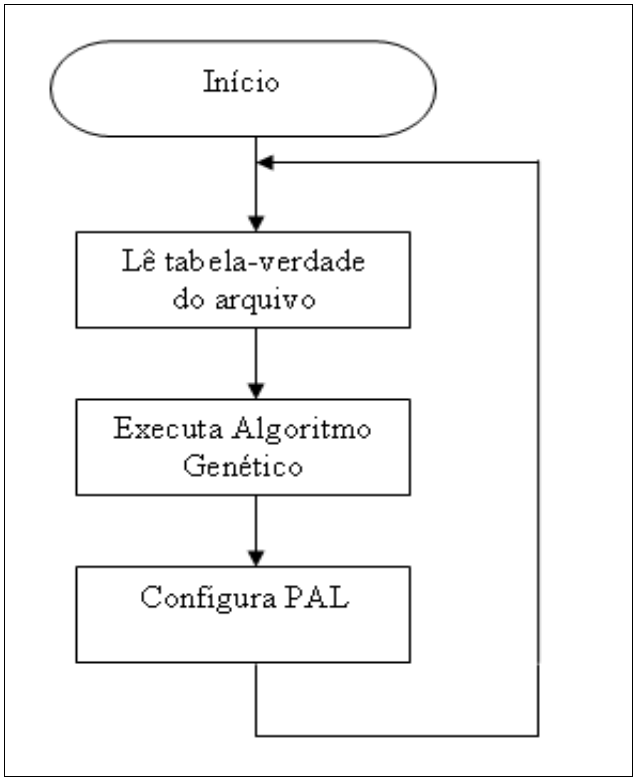


Figura 6.3: Fluxograma do sistema

FONTE: Dados do Trabalho

6.5 O *Hardware* do Sistema

O *hardware* basicamente é composto por um processador, uma memória RAM, um display LCD para exibir informações e uma PAL para ser reconfigurada. Todas essas estruturas foram implementadas no kit de desenvolvimento da *Altera Corporation* já citado anteriormente. Um diagrama simplificado do *hardware* pode ser visto na Figura 6.4. Através da interface JTAG a tabela-verdade será lida pelo *software* que estará executando na CPU NIOS II, a CPU por sua vez será responsável por se comunicar com o *display* LCD e também por configurar a PAL cada vez que um novo circuito for encontrado pelo Algoritmo Genético. A PAL recebe entradas binárias e sua saída será determinada de acordo com o circuito que está configurado atualmente nela. A memória será uma DDR SDRAM que irá armazenar o aplicativo em execução e seus dados (para saber como incluir uma memória DDR SDRAM em algum sistema *Stratix II* com o processador NIOS II consulte o seguintes manuais: Altera Corporation (2006a), Altera Corporation (2006b), Altera Corporation (2008)).

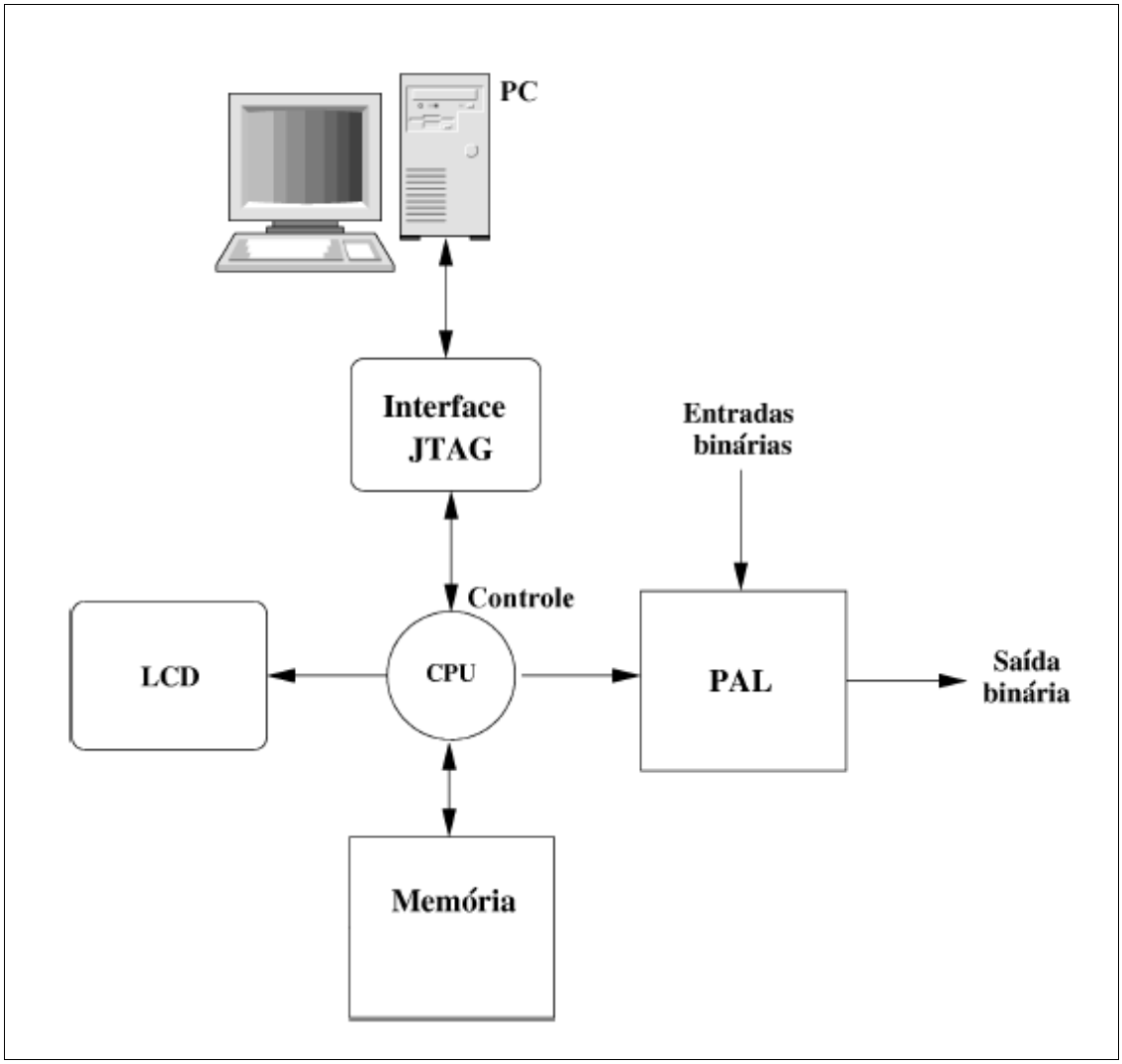


Figura 6.4: Hardware do Sistema

FONTE: Dados do Trabalho, adaptado de Lacerda (2006)

O diagrama de bloco do sistema inteiro foi dividido em partes para melhor visualização no texto. Na Figura 6.5 pode ser visto o diagrama de bloco do processador NIOS II utilizado. Seus pinos de entrada e saída estão conectados ou com outras partes do sistema ou com os pinos apropriados da FPGA como consta em Altera Corporation (2005).

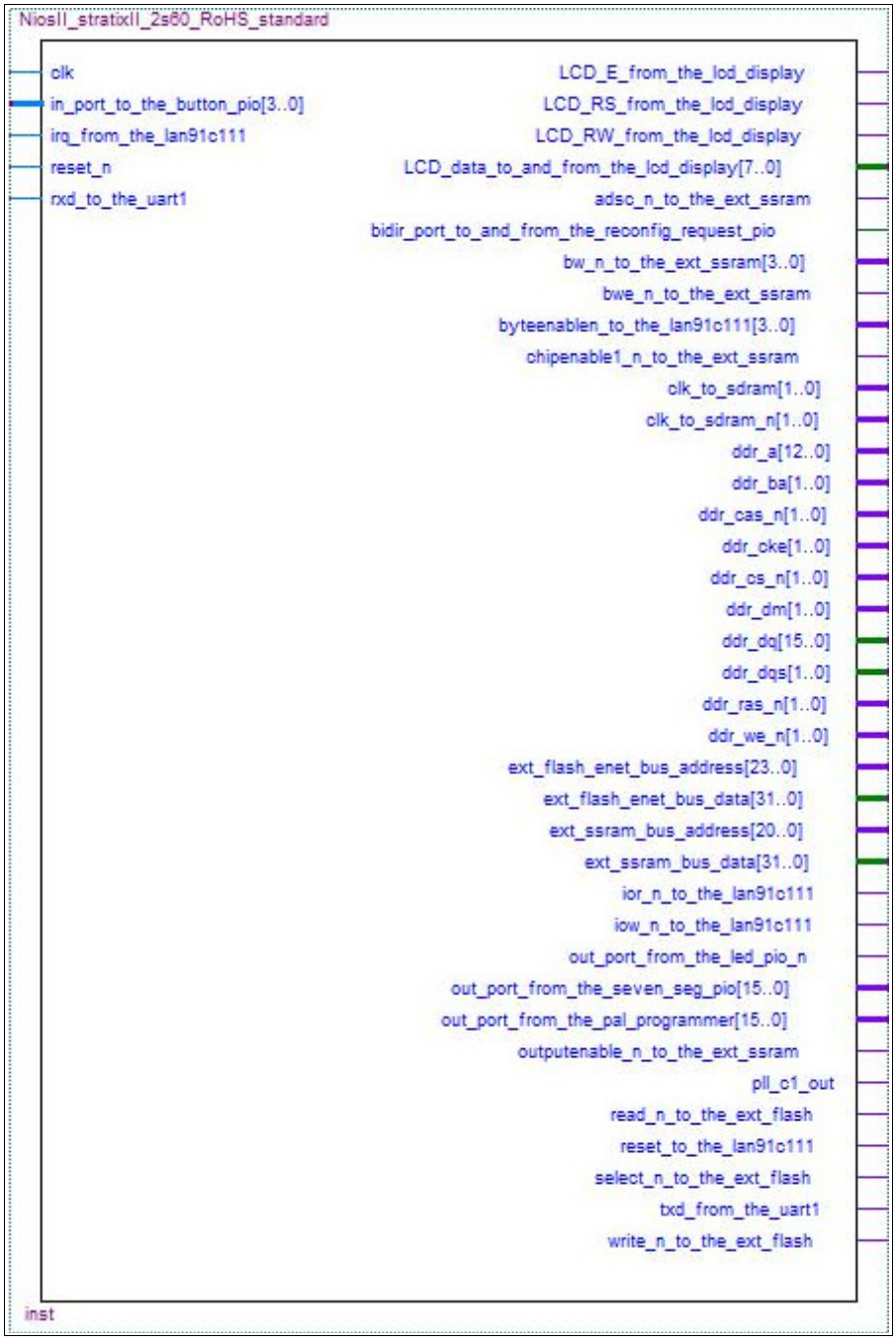


Figura 6.5: Diagrama de bloco do NIOS II gerado pelo *Quartus II*

FONTE: Dados do Trabalho

Na Figura 6.6 tem-se a maneira como os botões de entrada estão conectados aos LEDs, à entrada da PAL e aos pinos correspondentes na entrada do processador NIOS II (o mesmo nome foi dados aos componentes que estão interligados).

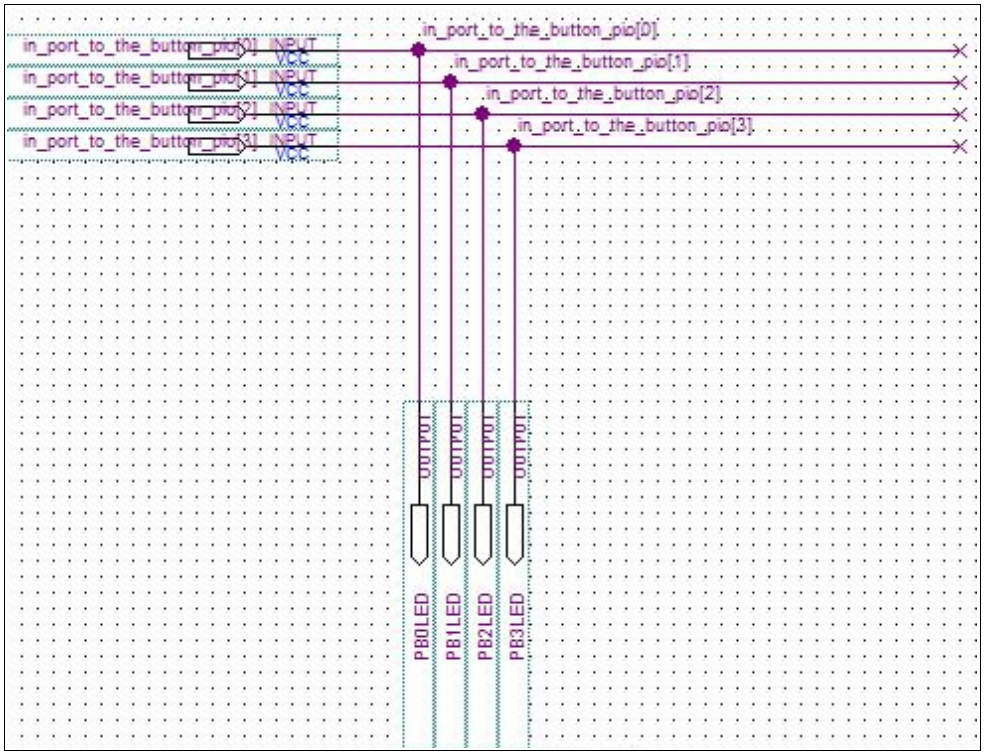


Figura 6.6: Diagrama de bloco dos botões de entrada da PAL

FONTE: Dados do Trabalho

E na Figura 6.7, tem-se o diagrama de bloco da PAL utilizada no sistema. A PAL recebe os dados de entrada através dos 4 botões presentes no kit, e a saída da PAL é indicada por um LED. Se a saída for 0 o LED se mantem apagado, se a saída for 1 o LED se mantem aceso. A PAL foi implementada na linguagem de descrição de *hardware* VHDL e foi integrada ao sistema através do diagrama de bloco no programa *Quartus II*.

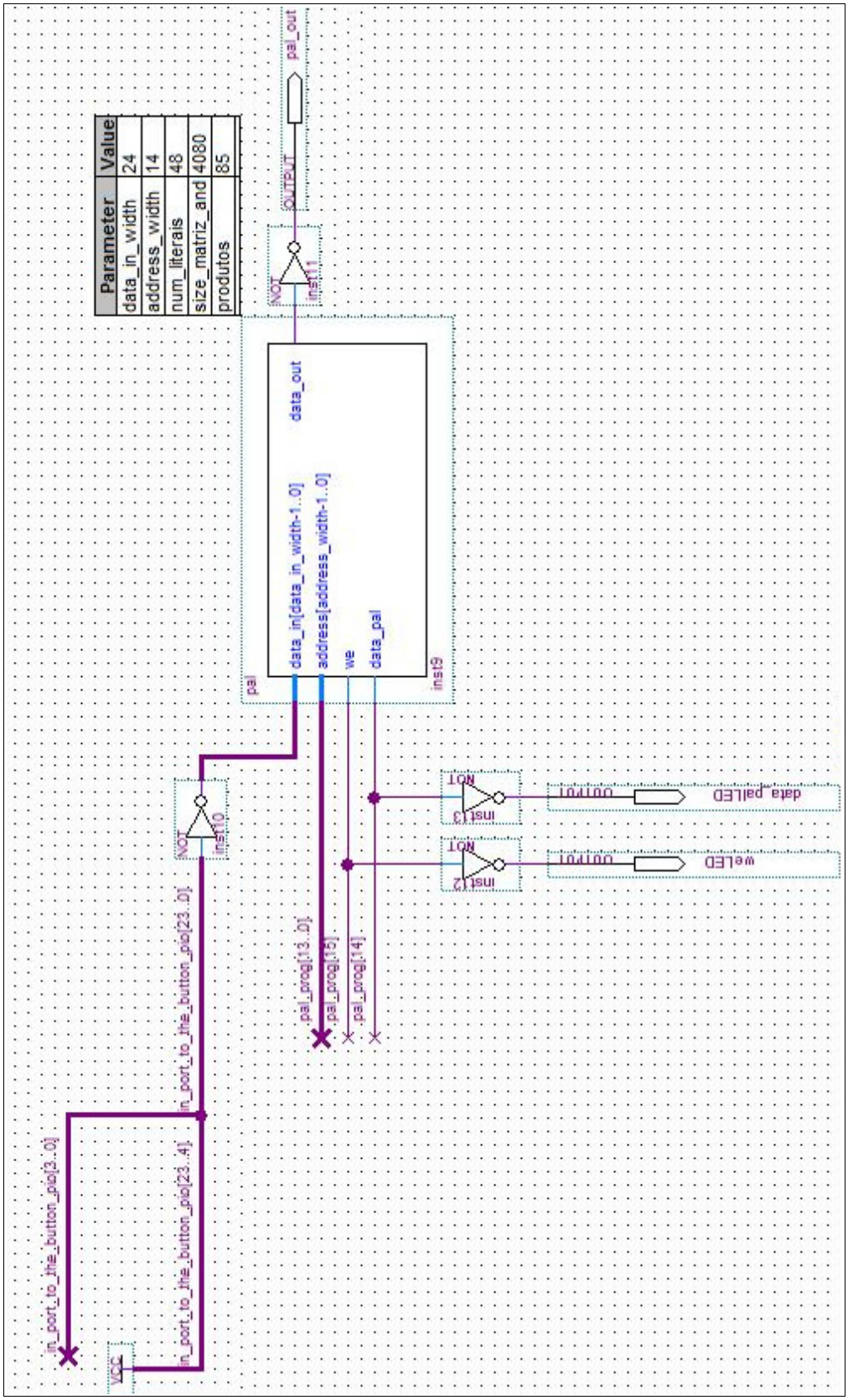
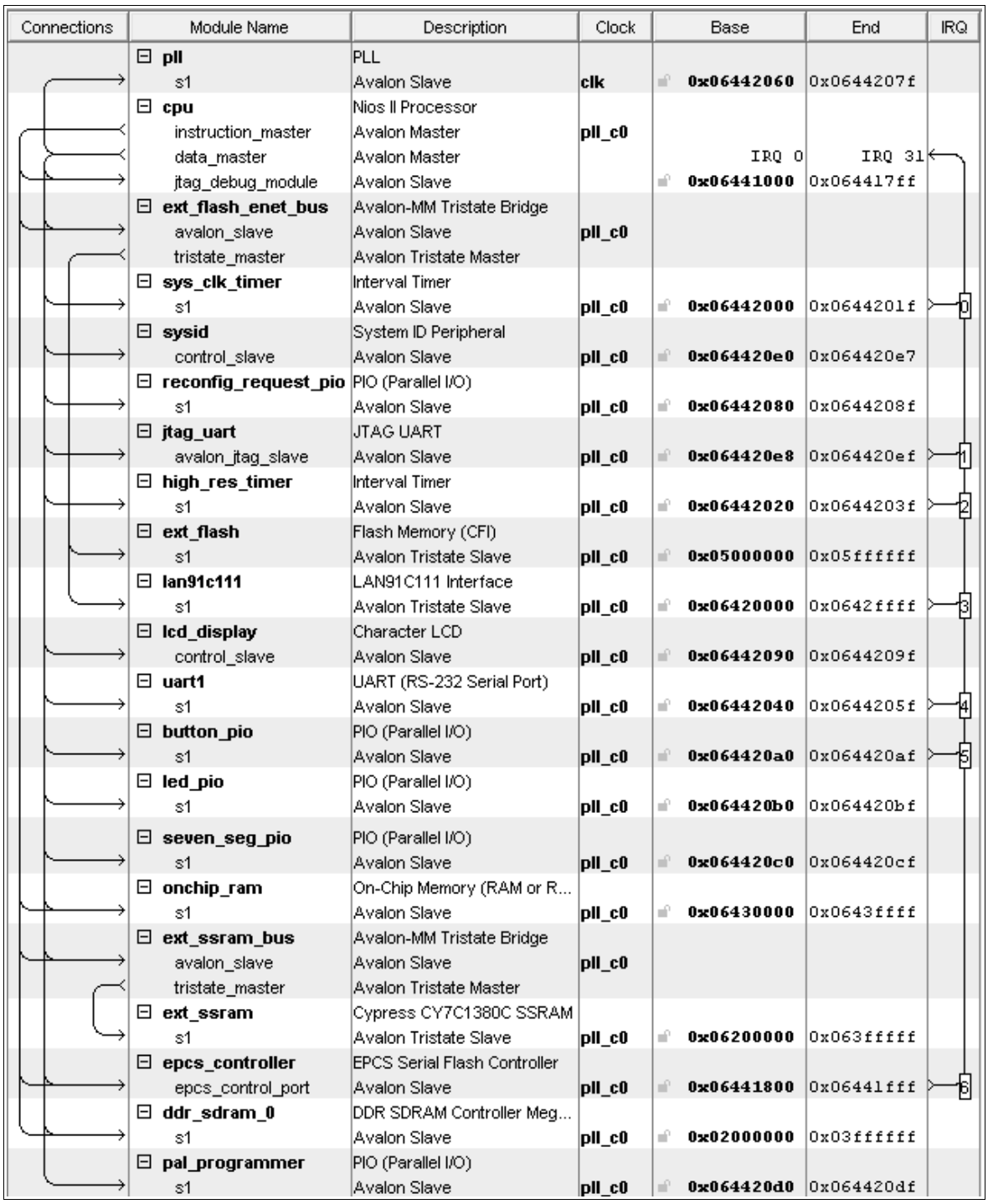


Figura 6.7: Diagrama de bloco da PAL

FONTE: Dados do Trabalho

Na Figura 6.8 podem ser observados os componentes do processador NIOS II no *SOPC Builder* (para saber como construir um sistema com o NIOS II no *SOPC Builder* consulte Altera Corporation (2007b) e Altera Corporation (2007c))**.**



Connections	Module Name	Description	Clock	Base	End	IRQ
	pll	PLL				
	s1	Avalon Slave	**clk**	**0x06442060**	0x0644207f	
	cpu	Nios II Processor				
	instruction_master	Avalon Master	**pll_c0**			
	data_master	Avalon Master		IRQ 0	IRQ 31	
	jtag_debug_module	Avalon Slave		**0x06441000**	0x064417ff	
	ext_flash_enet_bus	Avalon-MM Tristate Bridge				
	avalon_slave	Avalon Slave	**pll_c0**			
	tristate_master	Avalon Tristate Master				
	sys_clk_timer	Interval Timer				
	s1	Avalon Slave	**pll_c0**	**0x06442000**	0x0644201f	0
	sysid	System ID Peripheral				
	control_slave	Avalon Slave	**pll_c0**	**0x064420e0**	0x064420e7	
	reconfig_request_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**pll_c0**	**0x06442080**	0x0644208f	
	jtag_uart	JTAG UART				
	avalon_jtag_slave	Avalon Slave	**pll_c0**	**0x064420e8**	0x064420ef	1
	high_res_timer	Interval Timer				
	s1	Avalon Slave	**pll_c0**	**0x06442020**	0x0644203f	2
	ext_flash	Flash Memory (CFI)				
	s1	Avalon Tristate Slave	**pll_c0**	**0x05000000**	0x05ffffff	
	lan91c111	LAN91C111 Interface				
	s1	Avalon Tristate Slave	**pll_c0**	**0x06420000**	0x0642ffff	3
	lcd_display	Character LCD				
	control_slave	Avalon Slave	**pll_c0**	**0x06442090**	0x0644209f	
	uart1	UART (RS-232 Serial Port)				
	s1	Avalon Slave	**pll_c0**	**0x06442040**	0x0644205f	4
	button_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**pll_c0**	**0x064420a0**	0x064420af	5
	led_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**pll_c0**	**0x064420b0**	0x064420bf	
	seven_seg_pio	PIO (Parallel I/O)				
	s1	Avalon Slave	**pll_c0**	**0x064420c0**	0x064420cf	
	onchip_ram	On-Chip Memory (RAM or R...				
	s1	Avalon Slave	**pll_c0**	**0x06430000**	0x0643ffff	
	ext_ssram_bus	Avalon-MM Tristate Bridge				
	avalon_slave	Avalon Slave	**pll_c0**			
	tristate_master	Avalon Tristate Master				
	ext_ssram	Cypress CY7C1380C SSRAM				
	s1	Avalon Tristate Slave	**pll_c0**	**0x06200000**	0x063fffff	
	epcs_controller	EPCS Serial Flash Controller				
	epcs_control_port	Avalon Slave	**pll_c0**	**0x06441800**	0x06441fff	6
	ddr_sdram_0	DDR SDRAM Controller Meg...				
	s1	Avalon Slave	**pll_c0**	**0x02000000**	0x03ffffff	
	pal_programmer	PIO (Parallel I/O)				
	s1	Avalon Slave	**pll_c0**	**0x064420d0**	0x064420df	

Figura 6.8: Componentes do processador NIOS II no SOPC Builder

FONTE: Dados do Trabalho

O sistema completo ocupou apenas 16% dos elementos lógicos da FPGA e 45% dos pinos de E/S. Somente a PAL ocupou 11% dos elementos lógicos. A baixa eficiência na implementação da PAL se deve ao desperdício de elementos lógicos para implementar registradores de memória utilizados para armazenar o estado da matriz AND na PAL. Além disto, a PAL ocupa vários elementos lógicos para implementar as funções lógicas AND e OR.

6.6 O *Software* do Sistema

Esta seção irá descrever o *software* implementado no kit desde sua inicialização na leitura do arquivo de entrada até a gravação na PAL e seu reinício. O Fluxograma do *software* é o mesmo já apresentado na Figura 6.3. O programa executa três tarefas em *loop* infinito:

- Realiza leitura da tabela-verdade através de arquivo;
- Executa minimização do circuito através do Algoritmo Genético;
- Grava o circuito na PAL através de comandos enviados pela interface paralela do NIOS II.

6.6.1 Arquivos de Entrada

O *software* realiza a leitura de dois arquivos de texto: um que contem os parâmetros do Algoritmo Genético e o nome do arquivo que possui a tabela-verdade, e outro arquivo com a tabela-verdade propriamente dita.

O primeiro arquivo possui a estrutura mostrada na Figura 6.9. Na primeira linha do arquivo tem-se o número de gerações que o Algoritmo Genético irá executar. Na segunda linha, tem-se o número de indivíduos em uma população, taxa de cruzamento e taxa de mutação. E finalmente na terceira linha, tem-se o nome do arquivo que contem a tabela-verdade.

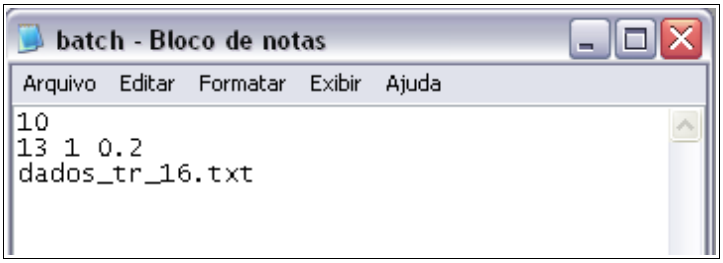


Figura 6.9: Arquivo com Parâmetros para o Algoritmo Genético

FONTE: Dados do Trabalho

O segundo arquivo, referente a tabela-verdade, possui uma estrutura semelhante a estrutura do arquivo de entrada do programa ESPRESSO como mostra a Figura 6.10. Logo após a *tag* ".i" tem-se o número de entradas da tabela. Após a *tag* ".o" tem-se o número de saídas da tabela. Depois da *tag* ".p" é indicado o número de linhas da tabela. Em seguida em cada linha do arquivo tem-se uma combinação das entradas e a saída desejada. E por fim, a *tag* ".e" indica o fim do arquivo.

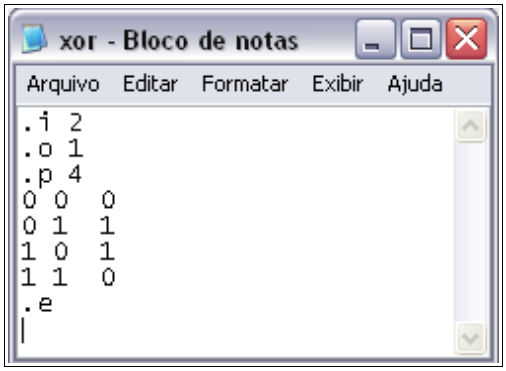


Figura 6.10: Arquivo com descrição da tabela-verdade

FONTE: Dados do Trabalho

6.6.2 O Algoritmo Genético

Para explicar como o problema foi modelado para ser resolvido pelo Algoritmo Genético serão utilizados dois indivíduos como exemplo (Figura 6.11) e a tabela-verdade da porta XOR da Figura 6.10.

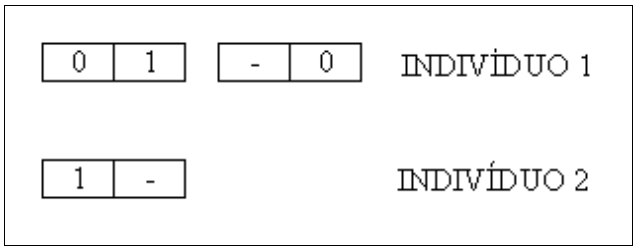


Figura 6.11: Indivíduos exemplo

FONTE: Dados do Trabalho

6.6.2.1 Codificação

Cada indivíduo é uma matriz onde cada linha representa um termo de produto e cada posição dessa linha representa uma entrada da tabela. Os valores possíveis para cada entrada são os seguintes:

- Valor da entrada negada (com operador NOT), representada pelo valor 0;
- Valor da entrada sem o operador NOT, representada pelo valor 1;
- Ausência da variável (*don't care*), representada no programa pelos valores 2 e 3 (foram utilizados dois valores para facilitar a mutação. A mutação será explicada mais adiante).

No exemplo da porta XOR cada linha da matriz possuirá duas posições já que a tabela possui duas entradas apenas e o número máximo de linhas para qualquer indivíduo será o número de linhas que a tabela possui cuja saída é 1, pois conforme explicado anteriormente sabe-se que o melhor circuito (o menor circuito possível em número de termos de produto) é sempre menor ou igual ao circuito no qual os termos de produto representam todas as linhas da tabela em que a saída é um.

Os indivíduos dados como exemplo representam as seguintes expressões *Booleanas* respectivamente (considerando que a primeira entrada é A e a segunda é B):

- $(\neg A \wedge B) \vee \neg B$;
- A .

6.6.2.2 Geração da População Inicial

A população inicial é gerada aleatoriamente mas baseada nos termos da tabela cuja saída é um. Primeiramente, sorteia-se quantos termos de produto cada indivíduo terá. Depois, são escolhidos aleatoriamente os termos de produto cuja saída é um para comporem a base do indivíduo, por exemplo, na tabela da porta XOR poderia ser o seguinte termo 01 (segunda linha da tabela). Em seguida, é feita uma perturbação de forma aleatória nesse termo inserindo *don't care.* Assim, os seguintes termos poderiam surgir a partir desse:

- 01 (sem inserção de *don't care*);
- 0-;
- -1;

- --.

Cada termo sofre apenas uma das perturbações possíveis (ou pode não sofrer nenhuma). Os termos resultantes após a perturbação são os termos que irão compor de fato o indivíduo. O *don't care* que surgiu no lugar de algum zero é representado pelo 2, e o que surgiu no lugar do 1 é representado pelo 3. Essa medida foi tomada para tornar o processo de perturbação reversível e principalmente para que a mutação (explicada mais abaixo) não gere perturbações totalmente aleatórias.

Está técnica faz com que os termos gerados para os indivíduos sejam sempre variações simples dos implicantes da função que se deseja otimizar descrita na tabela-verdade, isso faz com que o Algoritmo Genético não gere termos de produto muito distantes dos implicantes da função.

6.6.2.3 Estrutura da População

A estrutura da população utilizada foi inspirada na mesma utilizada em Toledo (2005) para o Problema Conjunto de Dimensionamento de Lotes e Programação da Produção. Outra estrutura foi testada no presente trabalho, mas como será mostrado na seção de resultados, a que obteve melhores resultados foi esta apresentada nesta seção.

A população é estruturada em *clusters* representados em uma árvore ternária onde os pais são sempre melhores que seus filhos de modo que o melhor indivíduo é sempre o nó raiz da árvore. Cada subárvore com seu pai e seus respectivos filhos é um *cluster*, como pode ser visto na Figura 6.12. Nessa figura tem-se uma população de 13 indivíduos: os *clusters* 1, 2, 3 e 4 são formados pelos seguintes grupos de indivíduos respectivamente: (0, 1, 2, 3), (1, 4, 5, 6), (2, 7, 8, 9) e (3, 10, 11, 12). Esta estrutura foi utilizada pois o método de seleção aplicado trabalha com os *clusters* como será explicado mais adiante.

Após a execução de uma geração, a população é reestruturada para manter essas características mencionadas acima.

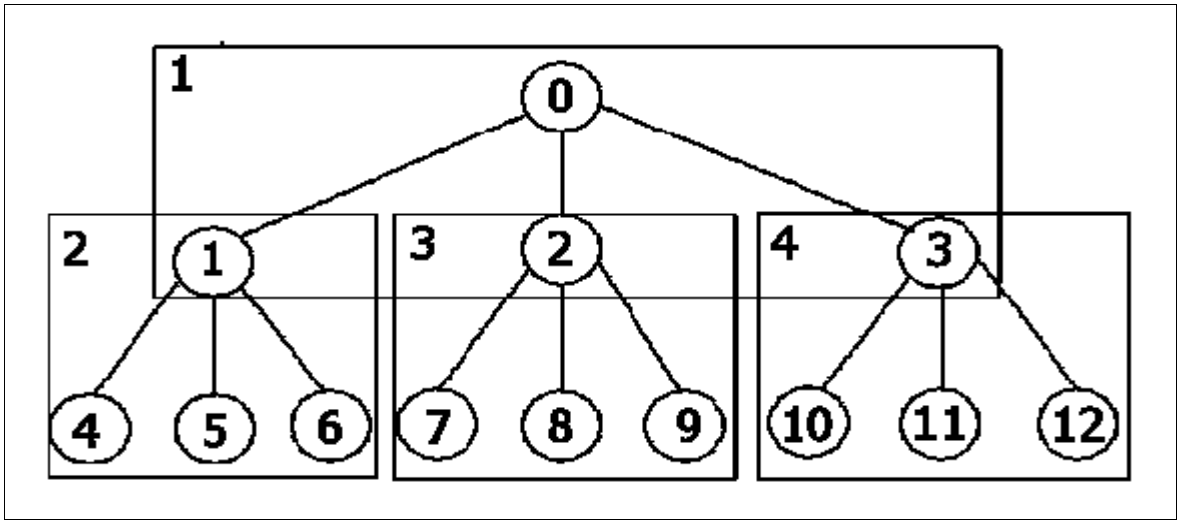


Figura 6.12: Estrutura da População

FONTE: Dados do Trabalho

6.6.2.4 Seleção

A seleção escolhe dois indivíduos que poderão ou não participar do cruzamento de acordo com a probabilidade. O primeiro indivíduo escolhido é sempre um pai de algum dos *clusters,* essa escolha é feita de forma aleatória. Logo em seguida é escolhido aleatoriamente um filho pertencente a este mesmo *cluster.* Dessa forma são escolhidos todos os sobreviventes para a próxima geração.

6.6.2.5 Cruzamento

O cruzamento implementado foi o chamado *cruzamento uniforme* envolvendo dois indivíduos que funciona da seguinte maneira:

É feita uma varredura passando por todos os termos de produto dos dois indivíduos pais até que se chegue ao fim do menor indivíduo pai, lembrando que o tamanho dos indivíduos é variável. Para cada termo de produto dos pais escolhe-se aleatoriamente se o filho receberá o termo do primeiro pai ou do segundo. Para as posições restantes do maior indivíduo pai é feito um sorteio para decidir qual desse termos irão para o filho.

6.6.2.6 Mutação

Foram implementados três tipos de mutações diferentes escolhidas aleatoriamente dentro do método da mutação:

- Realiza um *swap* em um valor de uma posição de acordo com a taxa de mutação, no caso os valores de troca são os seguintes: se o valor original for 0 troca-se por 2 e vice-versa; se o valor original for 1 troca-se por 3 e vice-versa. Essa medida foi tomada para que os termos gerados após sofrerem esse tipo de mutação não estejam tão distantes dos implicantes da função descrita pela tabela-verdade que são um guia para encontrar o circuito mínimo;

- Insere um novo termo de produto gerado aleatoriamente da mesma forma como os termos da população inicial são gerados (baseado nas linhas da tabela cuja saída é 1);

- Remove um termo de produto escolhido aleatoriamente.

6.6.2.7 Operador de Epidemia

O operador de epidemia simplesmente ao final de cada geração, mata quase todos os indivíduos e produz novos indivíduos aleatoriamente. O único indivíduo que sobrevive a epidemia é o melhor indivíduo da população.

Este operador tem a intenção de evitar convergência prematura e sempre manter uma população bem diversificada.

6.6.2.8 Função de Avaliação

A função de avaliação possui dois objetivos:

- Valorizar os indivíduos que acertam mais linhas da tabela-verdade;

- Valorizar os indivíduos menores.

Pode-se perceber que esses dois objetivos são muitas vezes conflitantes. Uma boa função de avaliação deve ser capaz de ponderar esses dois objetivos.

Para atender ao primeiro objetivo, a função utilizada foi a mesma encontrada em Stomeo (2006), e é a seguinte:

$$f_1 = 100 \div 2^i \times \sum_{j=0}^{2^i-1} (1 - |x_j - d_j|) \quad (1)$$

onde i é o número de entradas do sistema, x é a saída obtida pelo indivíduo e d é a saída desejada.

Para atender ao segundo objetivo utilizou-se a seguinte expressão:

$$f_2 = 1 \div (100 + tam) \quad (2)$$

onde *tam* é o número de termos de produto do indivíduo. A constante somada no denominador foi obtida através de testes, foi a constante que proporcionou melhores resultados e pareceu ponderar adequadamente o valor relativo entre os objetivos.

O *fitness* do indivíduo avaliado é:

$$f = -(f_1 + f_2) \quad (3)$$

e o objetivo do Algoritmo Genético é encontrar o indivíduo com o menor valor de *fitness*.

6.6.3 Critérios de Parada

Foram utilizados dois critérios de parada para o Algoritmo Genético usados em conjunto:

- Número máximo de gerações, passado pelo primeiro arquivo de entrada;
- O outro critério é finalizar o algoritmo pela convergência do melhor indivíduo, feita usando os valores do desempenho do melhor indivíduo da geração atual $f_{maxAtual}$ e da geração anterior $f_{maxAnterior}$, veja a equação abaixo:

$$f_{maxAtual} - f_{maxAnterior} = 0 \quad (4)$$

Se a situação descrita pela equação (4) ocorrer por *n* gerações consecutivas o Algoritmo Genético para sua execução. Onde *n = número de linhas da tabela-verdade * número de entradas da tabela-verdade*.

O Algoritmo Genético para sua execução quando qualquer um desses dois critérios é satisfeito.

6.6.4 Gravação na PAL e Reinício da Execução

Após o melhor indivíduo ser encontrado pelo Algoritmo Genético, ele é passado para o procedimento que o gravará na PAL. Depois do processo de gravação, a PAL estará pronta para uso. O programa então reinicia sua execução lendo novamente os arquivos de entrada, que

podem ter sido mudados simulando uma alteração do ambiente, uma mudança de necessidade. E assim se define o ciclo de execução do *software* do sistema.

7 RESULTADOS E DISCUSSÕES

Vários testes foram realizados para validar o sistema proposto e verificar os resultados. A seguir serão descritos alguns testes e também serão mostrados os resultados obtidos para algumas instâncias do problema.

7.1 Testes de *Hardware*

Para validar o *hardware*, testes funcionais foram realizados:

- Controle dos LEDs, botões e *display* LCD pelo processador NIOS II;
- Controle do envio de dados pela interface paralela destinada a comunicação com a PAL;
- Simulações do processo de gravação e funcionamento da PAL realizados no simulador existente no programa *Quartus II*.

Testes de integração do *hardware* foram feitos com ajuda do *software*, verificando por exemplo se os dados enviados pelo processador pela interface paralela de fato estavam chegando até a PAL. Pequenos programas foram feitos para acender os LEDs se algum botão fosse pressionado. Foram acrescentados no projeto do *hardware* LEDs em paralelo com a entrada dos comandos de escrita e dados da PAL**,** dessa forma pôde-se ver que quando os LEDs acendiam era porque de fato a PAL estava recebendo os dados enviados pelo processador NIOS II.

Esses testes apesar de simples são suficientes para constatar que o *hardware* e seus componentes funcionam e estão integrados. O *hardware* do sistema passou com sucesso nesses testes.

7.2 Testes de *Software*

Os testes realizados no *software* tiveram seu foco principal no módulo responsável por gerar o circuito a partir da tabela-verdade, ou seja, o módulo do Algoritmo Genético.

Entre os *benchmarks* mais famosos para testar a classificação binária da tabela-verdade está o teste da porta XOR que é capaz de mostrar que o sistema é capaz de resolver problemas de classificação não-linearmente separáveis. De fato, o Algoritmo Genético foi capaz de encon-

trar o circuito referente a porta XOR através de soma de produtos. Outros testes foram realizados como a porta OR, AND, NAND e NOR. Circuitos como do somador binário completo foram evoluídos. Tabelas arbitrárias com até quatro entradas foram testadas e seus resultados foram comparados com os resultados do ESPRESSO e foram os mesmos para o Algoritmo Genético proposto como pode ser visto na Tabela 7.2. Esta tabela mostra qual a tabela-verdade submetida a cada um dos métodos: AG1 – Algoritmo Genético mais simples (descrito logo abaixo), AG2 – Algoritmo Genético implementado como descrito no Capítulo 6, ESPRESSO – programa minimizador citado em capítulos anteriores. A tabela também mostra qual foi o termo de produto obtido em uma execução de cada um desses algoritmos. Uma tabela-verdade de 12 entradas foi usada para testes também, porém os Algoritmos Genéticos possuíram um desempenho bem pior se comparado ao ESPRESSO.

O Algoritmo Genético mais simples foi implementado a princípio, mas não obteve resultados satisfatórios. Esse Algoritmo Genético mais simples possui as seguintes diferenças:

- População não é estruturada em árvore, sua estrutura é simplesmente um vetor de indivíduos sem hierarquias;
- Não utiliza operador de epidemia;
- O método de seleção implementado é o Torneio.

As outras características são exatamente as mesmas do AG2.

Além dos resultados da Tabela 7.2 que mostram que o AG1 para tabelas um pouco mais complexas encontra circuitos maiores e menos eficientes que os outros métodos, a Tabela 7.1 mostra que o AG1 também é mais lento que o AG2. A tabela mostra o número de gerações que demorou para que cada algoritmo encontrasse o ótimo para tabelas com o número de entradas e número de linhas apresentados.

Tabela 7.1: Comparação entre os Algoritmos Genéticos

TABELA-VERDADE		Nº de Gerações - AG1	Nº de Gerações - AG2
Nº de Entradas	Nº de linhas		
2	4	10	9
3	8	61	43
4	16	111	70
8	27	356	309
12	171	11658	3230

Tabela 7.2: Comparação ESPRESSO - Algoritmos Genéticos

TABELA-VERDADE		SAÍDA - AG1	SAÍDA - AG2	SAÍDA - ESPRESSO
Entradas	Saída			
0 0	0	10	10	10
0 1	1	01	01	01
10	1			
11	0			
0 0	0	1-	1-	1-
0 1	1	-1	-1	-1
1 0	1			
1 1	1			
0 0	0	11	11	11
0 1	0			
1 0	0			
1 1	1			
0 0	1	-0	-0	-0
0 1	1	0-	0-	0-
1 0	1			
1 1	0			
0 0	1	00	00	00
0 1	0			
1 0	0			
1 1	0			
0 0 0	0	111	001	001
0 0 1	1	100	100	010
0 1 0	1	001	010	100
0 1 1	0		111	111
1 0 0	1			
1 0 1	0			
1 1 0	0			
1 1 1	1			
0 0 0 0	0	11-1	11-1	11-1
0 0 0 1	0	-111	-111	-111
0 0 1 0	0	1-11	1-1-	1-1-
0 0 1 1	0	1110		
0 1 0 0	0	1010		
0 1 0 1	0			
0 1 1 0	0			
0 1 1 1	1			
1 0 0 0	0			
1 0 0 1	0			
1 0 1 0	1			
1 0 1 1	1			
1 1 0 0	0			
1 1 0 1	1			
1 1 1 0	1			
1 1 1 1	1			

Para a realização dos testes acima, o Algoritmo Genético – AG2 – estava como descrito no Capítulo anterior e os valores de seus parâmetros eram os seguintes:

- Tamanho da população = 13 indivíduos;
- Taxa de cruzamento = 100%;
- Taxa de mutação = 20%.

Esses valores foram os mesmos assumidos pelo Algoritmo Genético mais simples – AG1.

7.3 Testes de Integração *Hardware/Software*

Após serem realizados os testes para o *hardware* e para o *software* separadamente e ser verificado que suas partes e seu todo estavam funcionando corretamente, o sistema foi integrado e seu funcionamento global pôde ser observado.

O programa executou continuamente sem problemas de acesso a memória e alocação dinâmica no kit. De fato a PAL era gravada após a execução do Algoritmo Genético no NIOS II e continuava funcionando enquanto o programa realizava outras tarefas como leitura do arquivo de entrada e evolução do circuito a ser gravado.

7.4 Limitações do Sistema e Propostas de Continuidade

O sistema proposto possui algumas limitações apesar de ter funcionado como esperado, são elas:

- O tempo de execução do Algoritmo Genético aumenta consideravelmente quando o número de entradas da tabela-verdade aumenta. Isso acontece porque quando se aumenta uma entrada o número de implicantes pode aumentar consideravelmente, o que causa um aumento também considerável no espaço de busca do problema;
- A PAL implementada suporta até 24 entradas, porém o *hardware* proposto utiliza apenas 4 entradas devido a quantidade de botões (quatro) do kit e do tempo de execução;

- Outro fator que amplifica o tempo de execução do Algoritmo Genético é que a função de avaliação é uma função caríssima, pois para cada indivíduo deve-se checar em toda a tabela-verdade o número de acertos;

- Encontrar uma função de avaliação que pondere corretamente os dois objetivos conflitantes se apresentou como uma dificuldade que só pôde ser resolvida empiricamente.

Várias soluções existem para minimizar essas questões apontadas acima e melhorar o sistema:

- Pode-se fechar o espaço de busca apenas nos implicantes primos, pois sabe-se que expressão *Booleana* que representa o circuito mínimo é formada por implicantes primos da função;

- Para melhorar o tempo de execução pode-se implementar operadores do Algoritmo Genético em *hardware*, ou acrescentar instruções específicas para realizar os operadores genéticos dentro da unidade aritmética lógica do NIOS II através do *SOPC Builder*;

- Implementar técnicas mais eficientes de avaliação para evitar percorrer novamente a tabela várias vezes para efetuar o mesmo cálculo, como por exemplo Programação Dinâmica;

- Implementar um *hardware* com vários processadores NIOS II para que o Algoritmo Genético possa ser multi-populacional e sua execução possa ser paralelizada, aumentando assim sua capacidade de explorar melhor o espaço de busca;

- Implementar um Algoritmo Genético multi-objetivo para conseguir otimizar de uma forma eficiente os dois objetivos conflitantes;

- Futuramente, podem ser acrescentadas melhorias como o sistema ser capaz de otimizar circuitos seqüenciais e não só combinacionais e o arquivo com a tabela-verdade poder conter mais de uma saída.

8 CONCLUSÃO

A metodologia apresentada neste trabalho mostra que é possível criar um sistema de *Hardware* Evolutivo com as tecnologias atuais disponíveis. Sistemas autônomos e dinâmicos podem ser construídos utilizando o processador NIOS II e o kit de desenvolvimento da *Altera Corporation*.

Essa metodologia viabiliza o estudo de várias técnicas de projeto de circuitos digitais e estudo de Algoritmos Genéticos com características diferentes e até outros métodos de aprendizado, uma vez que basta substituir a técnica do módulo que projeta o circuito do sistema pela técnica desejada. Outros experimentos podem ser realizados com outros tipos de *hardware* reconfiguráveis diferentes da PAL.

A FPGA é uma tecnologia extremamente interessante para o desenvolvimento de protótipos, pois facilmente se consegue realizar alterações e testes de *hardware* e além disso com custo bem reduzido.

A aplicabilidade do sistema proposto vai além da minimização de circuitos digitais. Várias aplicações podem ser estudadas já que EHW segundo a literatura pode ser usado também para reconhecimento de padrões.

Através do presente trabalho é possível perceber que o EHW é uma área muito promissora pois diminui o tempo de projeto do sistema, facilita mudanças no *hardware* (o que no *hardware* convencional não ocorre). EHW de fato aumenta o nível de automatização e pode-se construir sistemas *online* que sejam capazes de se adaptar a mudanças nas necessidades eliminando assim parte dos custos de manutenção e assistência técnica.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBERT, D.. Evolutionary Hardware Overview. **Johns Hopkins University Whiting School of Engineering**, Baltimore, Maryland, v. 1, n. 1, p. 1-20, 1997.

ALTERA CORPORATION. **Nios II Development Board, Stratix II Edition**. 1. ed. San Jose: Altera Corporation, 2005. 56 p.

ALTERA CORPORATION. **AN 398: Using DDR DDR2 SDRAM with SOPC Builder**. 1. ed. San Jose: Altera Corporation, 2006. 47 p.

ALTERA CORPORATION. **AN 327: Interfacing DDR SDRAM with Stratix II Devices**. 1. ed. San Jose: Altera Corporation, 2006. 52 p.

ALTERA CORPORATION. **Quartus II Handbook, Volume 4: SOPC Builder**. 1. ed. San Jose: Altera Corporation, 2007. 202 p.

ALTERA CORPORATION. **Nios II Hardware Development Tutorial**. 1. ed. San Jose: Altera Corporation, 2007. 50 p.

ALTERA CORPORATION. **Nios II Processor Reference Handbook**. 1. ed. San Jose: Altera Corporation, 2007. 232 p.

ALTERA CORPORATION. **DDR & DDR2 SDRAM Controller Compiler v 8.0 User Guide**. 1. ed. San Jose: Altera Corporation, 2008. 156 p.

ARAÚJO, S. G. de. **Síntese de Sistemas Digitais Utilizando Técnicas Evolutivas**. 2004. 138 p. Tese (Doutorado em Ciências em Engenharia Elétrica) - Universidade Federal do Rio de Janeiro COPPE/UFRJ, Rio de Janeiro.

BÄCK, T.; SCHWEFEL, H. P.. Evolutionary Computation: An Overview. **International Conference on Evolutionary Computation**, Nagoya, Japan, v. 1, n. 1, p. 20-29, 1996.

DARWIN, C.. **On the Origin of Species by Means of Natural Selection, or thePreservation of Favoured Races in the Struggle for Life**. 1. ed. London: John Murray, Albemarle Street, 1859. 247 p.

DE GARIS, H.. Genetic Programming: Artificial Nervous Systems Artificial Embryos and Embryological Electronics. **Parallel Problem Solving from Nature - Proceedings of 1st Workshop**, Springer-Verlag, Berlin, Germany, v. 496, n. , p. 117-123, 1991.

ERCEGOVAC, M.; LANG, T.; MORENO, J. H.. **Introdução aos Sistemas Digitais**. 1. ed. Porto Alegre: Bookman, 2000. 472 p.

GOLDBERG, D. E.. **Genetic Algorithms in search, optimization and machine learning**. 1. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. 372 p.

HIGUSHI, T.; IBA, H.; MANDERICK, B.. Evolvable Hardware. **Massively Parallel Artificial Inteligence**, Cambridge, MA: MIT Press, v. , n. , p. 398-421, 1994.

HIGUSHI, T.; IWATA, M.; KAJITANI, I.; IBA, H.; FURUYA, T.; MANDERICK, B.. Evolvable Hardware and its Applications to Pattern Recognition and Fault-tolerant Systems. **Toward Evolvable Hardware: The Evolutionary Engineering Approach**, Berlin, Germany: Springer-Verlag, v. 1062, n. , p. 118-135, 1996.

HIGUSHI, T.; IWATA, M.; KEYMEULEN, D.; SAKANASHI, H.; MURAKAWA, M.; KAJITANI, I.; TAKANAHASHI, E.; TODA, K.; SALAMI, M.; KAJIHARA, N.; OTSU, N.. Real-World Applications of Analog and Digital Evolvable Hardware. **IEEE Transactions on Evolutionary Computation**, , v. 3, n. 3, p. 220-235, Setembro 1999.

HUSSAIN, T. S.. An Introduction to Evolutionary Computation. **Tutorial presentation, 1998 CITO Researcher Retreat**, Hamilton, Ontario, v. 1, n. 1, p. 1-4, 1998.

JUNG, C. F.. **Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos.**. 1. ed. Rio de Janeiro - RJ: Axcel Books do Brasil Editora, 2004. 312 p.

LACERDA, W. S.. **Projeto e Implementação de Circuitos Classificadores Digitais com Controle da Generalização Baseado na Regra do Vizinho-mais-próximo Modificada**. 2006. 218 p. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal de Minas Gerais, Belo Horizonte.

LEVY, R.; LEPRI, S.; SANCHEZ, E.; RITTER, G.; SIPPER, M.. Slate of Art: An Evolving FPGA-Based Board for Handwritten-Digit Recognition. **Evolvable Hardware**, Palo Alto, CA, USA, v. , n. , p. 237-244, July 2000.

LINDEN, R.. **Algoritmos Genéticos**. 2. ed. Rio de Janeiro: Brasport, 2006. 428 p.

LOUIS, S. J.; RAWLINS, G. J. E.. Designer Genetic Algorithms: Genetic Algorithms in Structure Design. **Proceedings of the Fourth International Conference on Genetic Algorithms**, San Mateo, CA, v. , n. , p. 53-60, 1991.

SALAMI, M.; HENDTLASS, T.. The Fast Evaluation Strategy for Evolvable Hardware. **Genetic Programming and Evolvable Machines**, Netherlands, v. 6, n. 2, p. 139-162, 2005.

SILVA, A. C. R. da; BOVOLATO, M. C.; EMER, F. R. P.. Formulação do Problema de Cobertura de Funções Booleanas Através do Método de Expansão de Shannon. **World Congress on Engineering and Technology Education**, São Paulo - Brasil, v. 1, n. 1, p. 782-786, 2004.

SKLIAROVA, I.; FERRARI, A. B.. FPGA-based Implementation of Genetic Algorithm for the Traveling Salesman Problem and its Industrial Application. **Proceedings of the International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems- IEA/AIE'2002**, Cairns, Australia, v. 2358, n. 3, p. 77-87, Junho 2002.

SOARES, G. L.. **Algoritmos Genéticos: Estudo, Novas Técnicas e Aplicações**. 1997. 145 p. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Minas Gerais, Belo Horizonte.

STOMEO, E.. A Novel Genetic Algorithm for Evolvable Hardware. **IEEE Congress on Evolutionary Computation**, Vancouver, BC, Canada, v. , n. , p. 134-141, Julho 2006.

THOMPSON, A.. Silicon Evolution. **Genetic Programming 1996: Proc. 1st Annual Conf. (GP96)**, Cambridge, MA: MIT Press, v. , n. , p. 444-452, 1996.

TOCCI, R. J.; WIDMER, N. S.. **Sistemas Digitais: Princípios e Aplicações**. 8. ed. São Paulo: Pearson Prentice Hall, 2003. 768 p.

TOLEDO, C. F. M.. **Problema Conjunto de Dimensionamento de Lotes e Programação da Produção**. 2005. 141 p. Tese (Doutorado em Engenharia Elétrica e Computação) - UNICAMP, Campinas - SP.

YAO, X.; HIGUSHI, T.. Promises and Challenges of Evolvable Hardware. **IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews**, , v. 29, n. 1, p. 87-97, Fevereiro 1999.