UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA EMBARCADO RECONFIGURÁVEL DE FORMA ESTÁTICA POR PROGRAMAÇÃO GENÉTICA UTILIZANDO *HARDWARE* EVOLUCIONÁRIO HÍBRIDO

MANOEL ARANDA DE ALMEIDA

ORIENTADOR: PROF. DR. EMERSON CARLOS PEDRINO

São Carlos - SP
Fevereiro/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA EMBARCADO RECONFIGURÁVEL DE FORMA ESTÁTICA POR PROGRAMAÇÃO GENÉTICA UTILIZANDO *HARDWARE* EVOLUCIONÁRIO HÍBRIDO

MANOEL ARANDA DE ALMEIDA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Processamento de Imagens e Sinais.
Orientador: Dr. Emerson Carlos Pedrino.

São Carlos - SP
Fevereiro/2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Almeida, Manoel Aranda de
A447s Sistema embarcado reconfigurável de forma estática por programação genética utilizando hardware evolucionário híbrido / Manoel Aranda de Almeida. -- São Carlos : UFSCar, 2016.
94 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. Arquitetura virtual reconfigurável. 2. FPGA. 3. Hardware evolucionário. 4. GPLAB. 5. Programação genética. I. Título.

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Manoel Aranda de Almeida, realizada em 04/03/2016:

Prof. Dr. Emerson Carlos Pedrino
UFSCar

Prof. Dr. José Hiroki Saito
UFSCar

Profa. Dra. Adriane Beatriz de Souza Serapião
UNESP

Dedicatória: dedico este meu trabalho a minha esposa Olivia.

AGRADECIMENTO

Agradeço a Deus, que por amor tem preenchido minha vida com as mais belas oportunidades, ao meu pai (*in memoriam*) e a minha mãe que me mostraram o valor da educação e do trabalho, aos meus amigos, colegas e professores, em especial ao Professor Emerson Carlos Pedrino e meu colega Paulo Cesar Donizeti Paris, que em muito colaboraram com este meu desafio, e de uma forma muito especial a minha esposa Olivia, cujo apoio é fundamental para a jornada de minha vida.

A escuridão não pode expulsar a escuridão, apenas a luz pode fazer isso.

O ódio não pode expulsar o ódio, só o amor pode fazer isso.

Martin Luther King

RESUMO

O uso da tecnologia baseada em *Field Programmable Gate Arrays* (FPGAs), de forma reconfigurável, para a solução de diversos problemas atuais, tem se tornado um frequente objeto de estudo. Essa técnica é de aplicação viável e promissora na elaboração de sistemas embarcados, porém, a dificuldade em encontrar uma forma flexível e eficiente de realizar tal aplicação é o seu maior problema. Neste trabalho, é apresentada uma arquitetura virtual e reconfigurável (AVR) em FPGA para aplicações em *hardware*, utilizando um *software* de Programação Genética na elaboração de uma reconfiguração ótima para esta AVR, de forma a construir um *hardware* capaz de efetuar uma determinada tarefa em um sistema embarcado. Esta proposta é uma forma simples, flexível e eficiente de realizar aplicações adequadas em sistemas embarcados, quando comparada a outras técnicas de *hardware* reconfigurável. A representação do fenótipo no sistema evolutivo proposto se baseia em uma rede de elementos de função (EF) bidimensional. A ferramenta GPLAB, para MATLAB, é usada na Programação Genética, e a solução encontrada por esta é convertida em um mapeamento de memória com o cromossomo da melhor solução, onde este é usado para reconfigurar o *hardware*. Nos testes realizados, a GPLAB encontrou resultados para circuitos lógicos em poucas gerações, e para filtros de imagem encontrou soluções eficientes, onde ocorreu pouca ocupação de *hardware*, principalmente da memória nos casos apresentados, apresentando um cromossomo de tamanho reduzido, o que demonstra uma boa eficiência da proposta.

Palavras-chave: Arquitetura Virtual Reconfigurável. FPGA. *Hardware* evolucionário. GPLAB. Programação genética. Programação genética cartesiana. Sistemas embarcados. Programação de sistemas embarcados. Processamento Digital de Imagem.

ABSTRACT

The use of technology based on Field Programmable Gate Arrays (FPGAs), a reconfigurable technology, has become a frequent object of study. This technique is feasible and a promising application in the development of embedded systems, however, the difficulty in finding a flexible and efficient way to perform such an application is their bigger problem. In this work, a virtual and reconfigurable architecture (AVR) in FPGA for hardware applications is presented using a Genetic Programming Software on the development of an optimal reconfiguration for this AVR, in order to build a hardware capable of performing a given task in an embedded system. This proposal is a simple, flexible and efficient way to achieve appropriate applications in embedded systems, when compared to other reconfigurable hardware techniques. The representation of phenotype of the proposed evolutionary system is based on a bi-dimensional network function elements (EF). The GPLAB tool for MATLAB is used in Genetic Programming, and the solution found by this procedure is converted into a memory mapping to represent the best solution, where it is used to reconfigure the hardware. In the tests, GPLAB found results for logic circuits in a few generations, and for image filters containing efficient solutions, where there was little hardware occupation, especially memory, in the cases this has been presented, with a reduced chromosome size, shows a proposal efficiency.

Keywords: Virtual Reconfigurable Architecture. Field Programmable Gate Arrays. FPGA. Evolvable Hardware. GPLAB. Genetic programming. Cartesian genetic programming. Embedded systems. Programming embedded systems. Digital Image Processing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE – Algoritmo Evolutivo

CE – Computação Evolutiva

PG – Programação Genética

PGC – Programação Genética Cartesiana

PS – Processamento de Sinais

HE – *Hardware* Evolutivo

AVR – Arquitetura Virtual Reconfigurável

EF – *Elemento de Função*

ASIC – *Application-Specific Integrated Circuit*

CLB – *Configuration Logical Blocks*

FPGA – *Field Programmable Gate Array*

GP – *Genetic Program*

CGP – *Cartesian Genetic Program*

IC – Integrated Circuit

LUT – *Look-Up Table*

VLSI – *Very Large Scale Integration*

VRA – *Virtual Reconfigurable Architecture*

EHW– *Evolvable Hardware*

FE– *Function Element*

FIFO– *First In First Out*

MDPP– *Mean Difference perPixel*

DIP– *Digital Image Processing*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

O estudo da programação de sistemas embarcados para que produzam soluções mais adequadas, juntamente com técnicas de arquitetura virtual reconfigurável em FPGA e a programação genética cartesiana, são áreas de pesquisa que estão em expansão. Aqui é contextualizado o uso dessas técnicas na solução de problemas de programação de sistemas embarcados, utilizando a ferramenta GPLAB em um hardware reconfigurável.

1.1 Contexto

A Programação Genética (PG) é uma técnica automática de programação, que emprega os conceitos de Algoritmos Genéticos (AG) e Computação Evolutiva. Programação Genética e Algoritmos Genéticos são técnicas empregadas em casos de difícil solução por dedução analítica, pois executam uma busca otimizada em um conjunto amplo de possíveis soluções (PEDRINO, OGASHAWARA e RODA, 2010) (PEDRINO, RODA, *et al.*, 2011) (PEDRINO, 2008).

A GPLAB é uma *toolbox* de programação genética para MATLAB, que vem sendo utilizada para o estudo da programação genética; por ser versátil, generalista, de código livre, de uso livre e facilmente extensível, ela pode ser usada por todos os tipos de usuários, desde o leigo, ao pesquisador avançado (SILVA, 2007). Essa ferramenta contém várias aplicações que podem resolver problemas de regressão, entre outros, no entanto, não possui uma aplicação específica para gerar soluções para sistemas embarcados.

Sistemas embarcados estão presentes em praticamente todas as atividades diárias do ser humano. Esses sistemas realizam operações computacionais em

equipamentos que necessitam realizar tarefas de forma mais inteligente, e podem ser encontrados em aparelhos eletrônicos, vídeo *games*, máquinas fotográficas, geladeiras, aparelhos de micro-ondas, brinquedos, celulares, automóveis, etc. Nesse contexto, novas técnicas de programação que tornem a produção de sistemas embarcados mais eficiente, sempre serão bem-vindas.

O conceito de *hardware* evolutivo (HE) desenvolvido no início dos anos 1990, nas quais suas estruturas podem mudar e evoluir dinamicamente de forma autônoma por meio de interação com o ambiente; também mostrou ter um grande potencial no desenvolvimento de aplicações inovadoras (WANG, CHEN e LEE, 2007). Para essa evolução dinâmica, frequentemente são usadas técnicas de computação evolutiva.

HE tem sido explorado basicamente de duas formas. Na primeira, chamada HE extrínseca, experimentos simulados e evoluídos por *softwares* determinam o melhor cromossomo, e este é utilizado na configuração do *hardware*, portanto, configurado apenas uma vez; entende-se por cromossomo, um genótipo que descreve um determinado indivíduo. Na segunda, chamada de HE intrínseca, são utilizados dispositivos reconfiguráveis e o processo de obter o melhor cromossomo é realizado pelo próprio dispositivo, avaliando cada indivíduo candidato em seu próprio *hardware* (CANCARE, BHANDARI, *et al.*, 2011), podendo-se assim, explorar características típicas da plataforma usada, como temperatura ou níveis de tensão, no cálculo do valor de aptidão; essa segunda opção se mostra promissora devido ao surgimento e desenvolvimento de dispositivos lógico-reconfiguráveis, como, por exemplo, as FPGAs. Porém, nessas duas técnicas existem restrições; na primeira, um único circuito é obtido e deve ser confeccionado para realizar os testes práticos, onde isso gera um tempo de desenvolvimento muito grande, e na segunda, como a rotina da programação genética está embutida no circuito, grande parte dos recursos da FPGA é consumindo, sendo essa rotina de PG mais simples, logo, tal técnica também é muito pouco flexível, uma vez que o número de nós é fixo.

Na base de dados do IEEE (XPLORE, IEEE, 2014), têm-se 89 documentos referentes à programação genética cartesiana, e a GPLAB não é utilizada em nenhum deles; na base de dados da Scopus (SCOPUS, 2014), têm-se 181 documentos referentes à programação genética cartesiana, e a ferramenta GPLAB também não é utilizada.

Assim, nesta dissertação é proposta uma aplicação da ferramenta GPLAB, capaz de gerar soluções em um formato cartesiano e aplicáveis diretamente em sistemas embarcados utilizando AVR em um dispositivo de FPGA.

São utilizados os recursos da ferramenta para desenvolver a aplicação proposta, executar esta, adquirir novas soluções e realizar testes em dois problemas distintos; o primeiro é a solução para Circuitos Lógicos e o segundo trata da geração de filtros de processamento digital de imagens.

É proposta uma AVR com uma plataforma HE extrínseca e mais flexível. Essa arquitetura HE híbrida utiliza algumas qualidades das duas técnicas mencionadas, permitindo um alto desempenho de *hardware*.

A proposta é utilizar Elementos de Função (EF) montados em forma de matriz para reconfigurar a FPGA com o uso de linguagem de descrição de *hardware* (*Verilog Hardware Description Language* - *Verilog HDL*), permitindo o reuso do código em diferentes plataformas de FPGA, gerando-se assim, uma representação fenótipo/genótipo com menor custo, uma vez que a matriz de elementos é flexível.

EF's são elementos que podem ser configurados para assumir uma determinada função básica dentro de um conjunto de funções pré-estabelecidas, onde tais funções são combinadas dentro da matriz de forma que possam solucionar um determinado problema.

Para demonstrar a flexibilidade e eficiência da proposta são executadas duas simulações, sendo: uma para gerar circuitos lógicos para determinadas relações entrada/saída, e outra para gerar filtros de imagem digitais para filtrar ruídos dos tipos Sal e Pimenta e Gaussiano. Assim, foram obtidos resultados positivos em aplicações de diferentes complexidades.

Para a simulação de circuitos lógicos, a aplicação é executada 30 vezes a aplicação para determinar circuitos contendo três tipos de soluções pré-determinadas, onde os dados são registrados e coletados para determinar em qual geração ocorreu uma solução adequada e qual foi a eficiência do circuito.

Para a simulação de processamento digital de imagens, a aplicação é executada 100 vezes para filtragem, em tons de cinza, "lena" e "cameraman"; também, são registrados e coletados os dados dessas execuções para determinar em qual geração ocorreu uma solução adequada e qual é a eficiência em retornar para as imagens originais.

1.2 Motivações e Objetivos

A crescente necessidade de se criarem novos e mais adequados programas e circuitos para sistemas embarcados, de forma cada vez mais rápida, é um grande desafio na atualidade. O uso de HE é uma técnica promissora para o desenvolvimento de sistemas embarcados de maior eficiência, especialmente com o uso da AVR, bem como o uso da ferramenta GPLAB para auxiliar na criação desses sistemas, que em conjunto com HE, poderão ser viáveis para atender tal necessidade.

Pretende-se com este trabalho, demonstrar que a ferramenta GPLAB e a Arquitetura Virtual Reconfigurável (AVR) proposta podem ser usadas para criar esses sistemas, facilitando-se a utilização de programação genética na solução de problemas de *hardware* de sistemas embarcados.

1.3 Metodologia de Desenvolvimento do Trabalho

Neste Trabalho é utilizada a ferramenta GPLAB em conjunto com o MATLAB, para criar um conjunto de funções capazes de gerar resultados em formato cartesiano a serem empregados diretamente em um sistema embarcado.

As soluções para geração de Circuitos Lógicos e Filtros de Imagens são comparadas entre si e com o estudo de Vasu, Srivastava et al. (VASU, SRIVASTAVA, *et al.*, 2014) para solução de circuitos e de Wang, J., Chen, Q. S. e Lee, C. H. (WANG, CHEN e LEE, 2007) para solução de filtros.

1.4 Organização do Trabalho

No Capítulo 2 são abordados conceitos de computação evolucionária, seus componentes e algoritmos evolucionários, tais como, algoritmo genético, programação genética e programação genética cartesiana. Também, é apresentada a ferramenta GPLAB para MATLAB e como utilizar a mesma; são abordados, também,

os conceitos de processamento digital de sinais, morfologia matemática e seus operadores básicos, conceitos de sistemas embarcados e de *hardware* evolutivo.

No capítulo 3 é demonstrada a organização da Proposta e a metodologia utilizada e sua implementação; também, é demonstrado como a ferramenta GPLAB e AVR foram adaptadas para serem aplicadas em duas simulações diferentes; uma é a simulação de geração de circuitos lógicos e a outra a de filtros para processamento digital de imagens.

No Capítulo 4 são apresentados os resultados das simulações e suas comparações.

Por fim, no Capítulo 5 é apresentada a conclusão; aqui é demonstrado que a GPLAB e AVR são ferramentas adequadas e aplicáveis ao desenvolvimento de sistemas embarcados utilizando a arquitetura proposta.

Capítulo 2

FUNDAMENTAÇÃO

Neste capitulo é pretendido abordar os conceitos básicos sobre Computação Evolucionária, em especial a Programação Genética, a Programação Genética Cartesiana, a ferramenta GPLAB para MATLAB, o processamento digital de imagens, sistemas embarcados e hardware evolutivo.

2.1 Considerações Iniciais

A Programação Genética (PG) é uma técnica automática de programação que emprega os conceitos de Algoritmos Genéticos e Computação Evolutiva. Vale lembrar que o conceito de Programação Genética e Algoritmos Genéticos são técnicas empregadas em casos de difícil solução analítica, pois executam uma busca otimizada em um conjunto amplo de possíveis soluções.

Sistemas embarcados estão presentes em praticamente todas as atividades diárias do ser humano. Esses sistemas realizam operações computacionais em equipamentos que necessitam realizar tarefas de forma mais inteligente, e podem ser encontrados em aparelhos eletrônicos, *vídeo games*, máquinas fotográficas, geladeiras, aparelhos de micro-ondas, brinquedos, celulares, automóveis, etc.

A GPLAB é um conjunto de ferramentas de programação genética para MATLAB, que vem sendo utilizada para o estudo da programação genética.

Hardware Evolutivo (HE) é uma técnica que emprega os conceitos de sistemas evolutivos, utilizando algoritmos genéticos, programação genética, programação genética cartesiana, entre outras abordagens, para encontrar um *hardware* adequado a um problema que se deseja resolver.

2.2 Conceitos da Computação Evolucionária

Na natureza, entidades que são mais aptas a executar tarefas em seu ambiente sobrevivem e se reproduzem a uma taxa superior. Este é o conceito da sobrevivência do mais apto, denominado seleção natural e descrita por Charles Darwin em “A Origem das Espécies por Meio da Seleção Natural” (DARWIN, 1859).

A computação evolucionária (CE) utiliza modelos computacionais de processos evolucionários baseados na teoria de Darwin, e sendo assim, pode-se considerar que a computação evolucionária imita o processo da evolução natural, por meio de processos de seleção e reprodução (PEDRINO, 2008).

Em CE foram desenvolvidos algoritmos capazes de criar soluções adequadas a diversos problemas complexos, em muitos casos ainda não resolvidos adequadamente por outras técnicas computacionais (GABRIEL e DELBEM, 2014).

2.2.1 Algoritmos Evolucionários

Os métodos de Algoritmos Evolucionários (AEs) apresentam alguma simplicidade utilizando poucas linhas de código, baseadas nos princípios básicos da Teoria da Evolução e Genética, para encontrar essas soluções, além de apresentar solução para diversas áreas com uma fácil adaptação. Devido a essas qualidades, a CE tem se mostrado uma área de pesquisa em rápida expansão.

Os primeiros trabalhos abordando algum tipo de AE foram estudos de sistemas evolutivos naturais investigados como algoritmos de exploração de múltiplos picos de uma função objetivo, datados da década de 1930. A falta de plataformas computacionais impediu a evolução desses estudos. Porém, após algumas décadas houve uma retomada desses estudos com os trabalhos de Holland, Rechenberg, Schwefel e Fogel (HOLLAND, 1962), (RECHENBERG, 1964), (SCHWEFEL, 1981) e (FOGEL, 1962). Hoje em dia há um expressivo crescimento em publicações nessa área.

O conceito básico de CE consiste em se criar um conjunto de soluções possíveis, depois fazer uma avaliação dessas soluções, em seguida se criar uma forma de selecionar e combinar essas soluções, de maneira a gerar um novo conjunto de soluções e repetir essa interação até se conseguir uma solução ótima. Uma

representação geral de um AE típico pode ser vista no pseudocódigo do algoritmo (EIBEN e SMITH, 2003), de acordo com a Figura 2.1.

```
1 INICIALIZAR população com soluções candidatas aleatórias
2 AVALIAR cada candidata
3 REPETIR:
4 SELECIONAR pais
5 RECOMBINAR pares de pais
6 MUTAR os descendentes resultantes
7 AVALIAR novas candidatas
8 SELECIONAR indivíduos para a nova geração
9 até a CONDIÇÃO DE PARADA ser satisfeita;
```

Figura 2.1 Pseudocódigo do algoritmo evolucionário (EIBEN e SMITH, 2003).

Os AEs seguem processos evolutivos, semelhantemente ao que ocorre na natureza, e seus principais componentes são (JONG, 2008);

- População – uma ou mais coleção de indivíduos concorrem entre si.
- Aptidão – qualidade de um indivíduo quantificada por meio de uma métrica escolhida para determinar uma medida de avaliação.
- Seleção – processo de escolha de indivíduos de cada população para serem usados na reprodução.
- Reprodução – processo de recombinação dos indivíduos com o objetivo de se gerar novos indivíduos.

Esse processo pode ter muitas variantes, mas todos os algoritmos evolucionários (AE) seguem um mesmo processo básico: criar uma população de indivíduos, submeter os mesmos a uma pressão ambiental e selecionar os mais aptos para sobreviverem.

2.2.2 Componentes de Algoritmos Evolucionários

Os principais componentes dos algoritmos evolucionários são (RAYWARD-SMITH, OSMAR, *et al.*, 1996), (REEVES, 1993), (GALVÃO e VALENÇA, 1999):

- Representação – definição dos indivíduos e criação de um conjunto de soluções adequadas ao problema. Os objetos formando possíveis soluções do contexto do referido problema são chamados fenótipos e os indivíduos codificados do AE são chamados genótipos. A representação consiste num mapeamento de fenótipos num conjunto de genótipos

(PEDRINO, 2008). Então, genótipo é uma representação que codifica o fenótipo, por sua vez, o fenótipo é o indivíduo que é obtido por um genótipo e que está pronto para ser submetido ao processo de Aptidão.

- Aptidão – é uma função de avaliação que extrai uma medida de qualidade dos genótipos, indicando o quanto cada indivíduo é capaz de solucionar o problema.
- População de Indivíduos – é um conjunto de indivíduos ou as representações de possíveis soluções. Normalmente, a população tem tamanho fixo e não se altera durante o processo de evolução; geralmente, a primeira população é gerada de forma aleatória, entretanto, pode ser gerada de uma forma orientada, por exemplo, uniformemente em uma grade ou orientando para alguma região no espaço de busca.
- Mecanismos de Seleção de Pais – normalmente, são processos probabilísticos de seleção de indivíduos baseados em sua aptidão, ou seja, indivíduos são escolhidos por um determinado processo para a operação de variação. Assim os mais aptos terão maior probabilidade de serem selecionados para uma operação de variação.
- Operadores de Variação – ou operadores genéticos, realizam a combinação de indivíduos selecionados para criação de novos indivíduos. A variação pode ser sexuada (cruzamento), onde dois ou mais pais são combinados e geram um ou mais filhos, ou pode ser assexuada (mutação), onde um indivíduo selecionado é modificado em parte. Diferentes tipos de operadores genéticos estão descritos na literatura.
- Mecanismos de Substituição – processo que seleciona os indivíduos que devem permanecer na nova população após o processo de variação. Podem ser do tipo Geracional, Geracional com Elitismo e Regime Permanente; no primeiro, os pais são substituídos pelos filhos em parte ou totalmente; no segundo, um pequeno grupo de pais não é substituído, e no último, apenas um número restrito de pais é substituído.

- Condição de Parada – processo que identifica se houve uma solução ótima, mas, como não há garantia de que a solução ótima seja encontrada, identifica um momento de parar a execução.

Os mais conhecidos AEs são: a Programação Evolutiva, a Estratégia Evolutiva, os Algoritmos Genéticos e a Programação Genética, onde todos têm os mesmos princípios básicos, porém, são abordados apenas Algoritmos Genéticos e a Programação Genética neste texto, por serem objetos do nosso estudo.

2.2.3 Algoritmo Genético

O AG gera descendentes utilizando os operadores de variação: cruzamento e mutação, estabelecendo um equilíbrio entre preservar os melhores indivíduos e explorar o espaço de busca (GABRIEL e DELBEM, 2014).

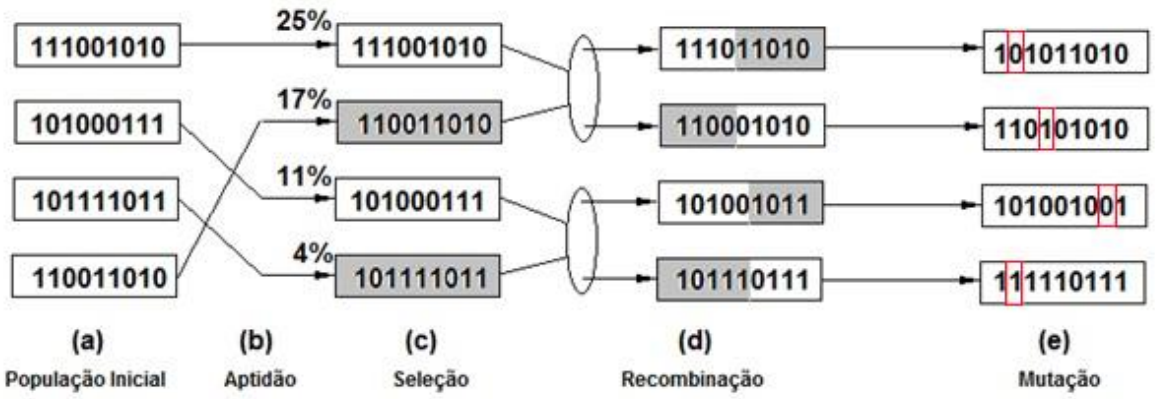

Figura 2.2 Processo de Recombinação e Mutação.
Adaptado de (MILLER, 2011).

A recombinação seleciona dois ou mais indivíduos, que passam a ser chamados de pais, e são combinados entre si gerando dois ou mais indivíduos, que passam a ser chamados de filhos. A mutação seleciona um indivíduo que sofre uma alteração. Esses novos indivíduos são avaliados pela aptidão e competem com os demais indivíduos para permanecerem na nova população por meio de um procedimento de seleção definido. Esse processo é repetido até que algum indivíduo seja capaz de solucionar o problema de forma ótima ou que seja atingido um determinado número de interações. Como podemos ver na Figura 2.2 em: (a) ocorre a geração da população inicial; (b) aplicada a métrica de aptidão; (c) classificação pela

seleção; (d) recombinação dos pais selecionados; e (e) mutação com alteração de parte do indivíduo.

Portanto, o AG faz uma busca dentro do espaço de todas as combinações possíveis de indivíduos e tende a localizar uma região onde existem indivíduos mais aptos.

2.2.4 Programação Genética

Em 1988, John Koza apresentou a programação genética sob a forma de patente, onde esta forma de AE utiliza um algoritmo genético não linear na solução de problemas (KOZA, 1992).

Há muitos problemas diferentes que exigem a descoberta de um programa de computador, ou uma rotina qualquer, que produz algum resultado desejado para entradas particulares. De acordo com Koza, o processo de resolução desses problemas pode ser reformulado como uma busca do melhor programa de computador a solucionar o problema, dentro do conjunto dos possíveis programas de computador.

PG difere de AG, pois suas soluções são programas de computador, compostos por rotinas capazes de efetuar diversas ações simples e necessárias ao programa, que combinadas vão executar uma determinada tarefa que se deseja.

Da mesma forma que um AG, uma população inicial de indivíduos é criada; nessa população, cada indivíduo é gerado sendo um programa, e cada um representa uma possível solução de um determinado problema; esse conjunto de indivíduos passa por uma função de avaliação de aptidão, onde são classificados; essa função é executada para cada indivíduo. As melhores soluções ou os melhores programas são selecionados para darem origem a uma nova população pela aplicação de operadores de recombinação e/ou mutação. Da mesma forma, essa interação é repetida até que uma condição de parada seja atingida, com a solução do problema, ou atingindo um número de interações definidas.

Os indivíduos são gerados aleatoriamente compostos por funções e terminais adequados para o domínio do problema. Como, na Figura 2.3, têm-se funções de aritmética (+ e *) e os terminais são as variáveis (A, B e C), formando um indivíduo para a solução de um problema (S=A + B * C).

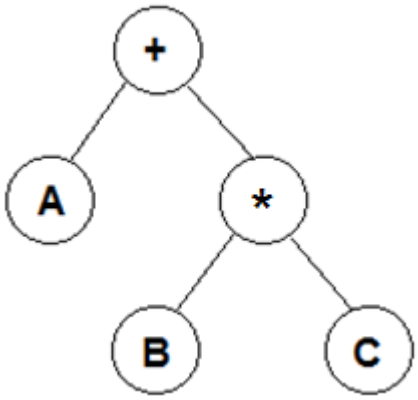

Figura 2.3 Indivíduo com Terminais e funções.

Pode-se utilizar qualquer tipo de função, como funções matemáticas, funções lógicas ou funções específicas. Nos terminais, podem-se utilizar variáveis de qualquer tipo: valores inteiros, valores reais, valores complexos, vetores, etc. Podem-se utilizar até mesmo programas ou fragmentos de programas de computador.

Os indivíduos são apresentados, normalmente, em forma de árvore.

Após a criação de uma população inicial de indivíduos, ou seja, um conjunto de programas de computador adequados ao problema, cada um desses programas de computador é executado no ambiente de simulação do problema e uma medida de aptidão é extraída. Essa medida de aptidão varia conforme o problema e deve indicar da melhor forma possível o quanto o programa é capaz de solucionar o problema.

Por exemplo, no caso simples de se tentar encontrar uma determinada solução matemática, a aptidão seria a soma dos módulos da subtração entre o valor desejado e o valor encontrado pelo programa do computador, e quanto mais próximo de zero for, melhor o programa soluciona a questão (SILVA, 2007).

Com essa população inicial enumerada pela aptidão, pode-se aplicar o princípio de reprodução e sobrevivência dos indivíduos mais aptos. Também, pode-se usar a recombinação genética sexuada, chamada de cruzamento, e em conjunto com a mutação, criar uma nova população. No caso de reprodução sexuada, dois ou mais programas pais são selecionados conforme sua aptidão e uma sub-árvore é escolhida aleatoriamente de cada um, onde são trocadas entre si, como demonstrado na Figura 2.4. No caso da mutação, uma função, um terminal ou uma sub-árvore é escolhida aleatoriamente e trocada por outra criada aleatoriamente, como na Figura 2.5.

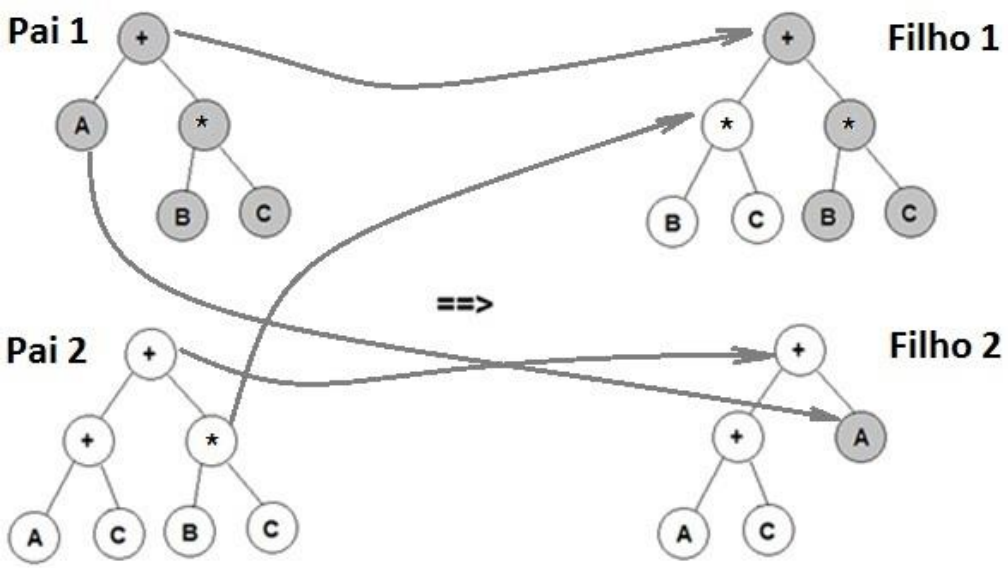

Figura 2.4 Recombinação Genética Sexuada.

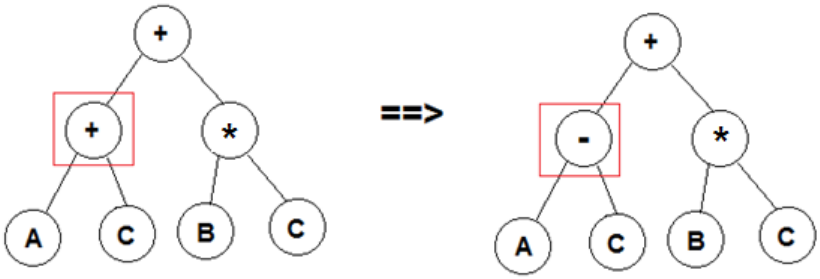

Figura 2.5 Mutação.

Assim, novos programas de computador são criados, e cada um é avaliado para extrair sua aptidão; uma etapa de seleção é então aplicada para selecionar quais indivíduos devem fazer parte da nova população, substituindo a população anterior. Essa nova população pode ou não conter indivíduos aptos da geração anterior, conforme a estratégia de seleção escolhida.

A repetição dessa interação de muitos indivíduos ao longo de várias gerações tende a produzir programas de computador mais aptos em resolver o problema proposto.

Essa interação é repetida até se atingir a condição de parada, ou seja, encontrar uma solução ou atingir um determinado número de repetições, como demonstrado pelo fluxograma da Figura 2.6.

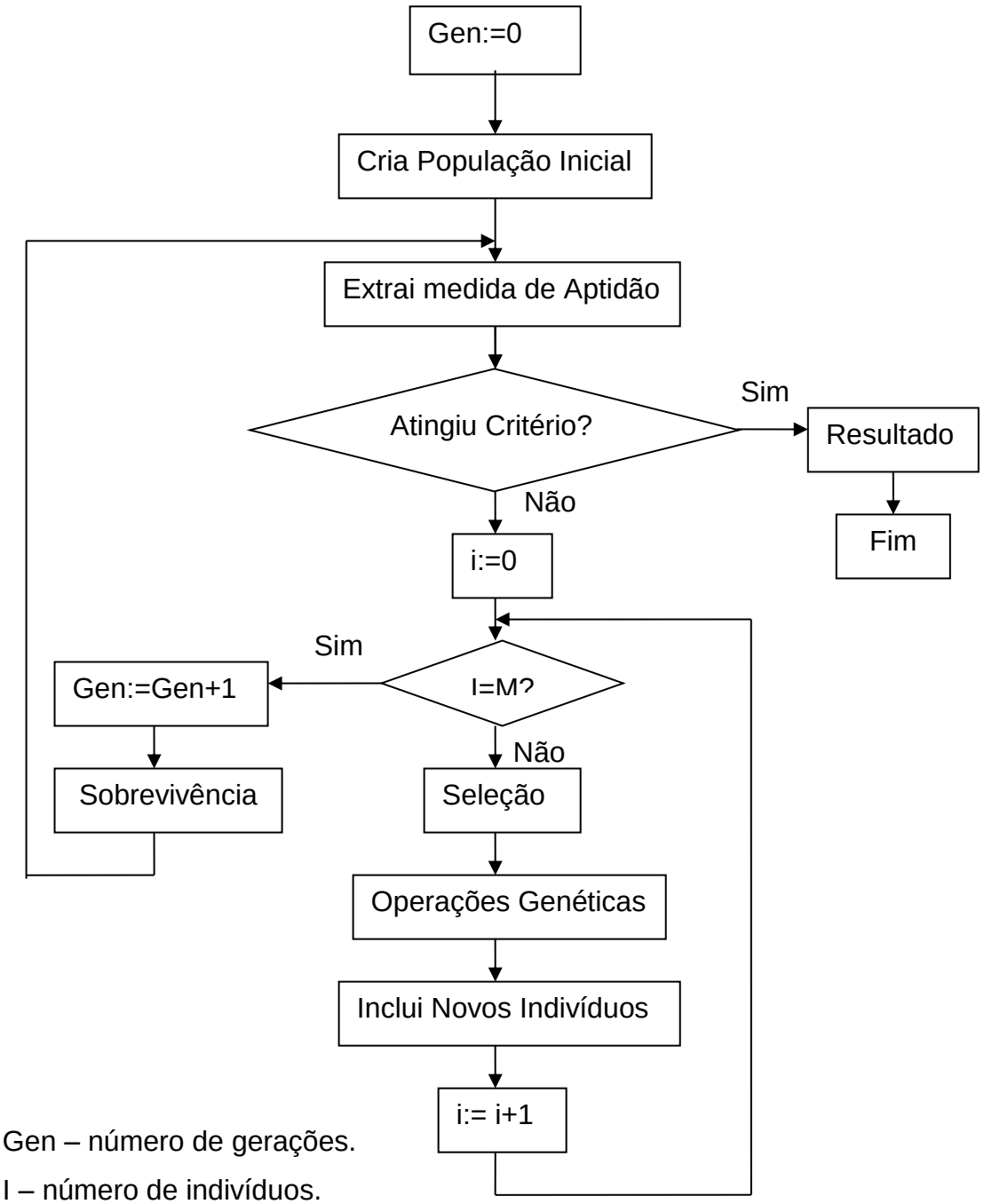

Figura 2.6 Fluxograma da Programação Genética.

2.2.5 Programação Genética Cartesiana

Apresentada pela primeira vez por J. Miller como um método para evolução de circuitos (MILLER, THOMSON e FOGARTY, 1997), onde somente mais tarde foi

usado o termo “Programação Genética Cartesiana” (MILLER, 1999). A Programação Genética Cartesiana (PGC) é um AE proposto como uma forma geral de programação genética (PG).

A programação genética cartesiana (PGC) apresenta o programa na forma de uma grade bidimensional de nós, onde um nó é qualquer operação computacional sobre os dados presentes em suas entradas. Entradas, saídas e saídas dos nós são enumerados sequencialmente por números inteiros. O cromossomo é uma representação do genótipo que contém as configurações do indivíduo em avaliação e é apresentado apenas como uma sequência linear de inteiros separados por espaços (MILLER, 1999).

Na Figura 2.7, pode-se ver um exemplo de programa com seu genótipo e fenótipo, para a implementação de dois conjuntos de operações lógicas de circuitos. As funções “AND” são identificadas como *1* e as funções “OR” são identificadas como *2*; as entradas são *0, 1, 2, 3, 4, 5*; as saídas são *10* e *11*. O genótipo descreve os circuitos lógicos que são implementados no fenótipo. Cada gene é uma sequência de inteiros que codifica um nó, e este é codificado em sequência como entrada1, entrada2, função e saída. Neste caso, cada grupo de quatro números separados por um espaço simples é um gene e o genótipo é composto por seis genes, que codificam cada um dos seis nós.

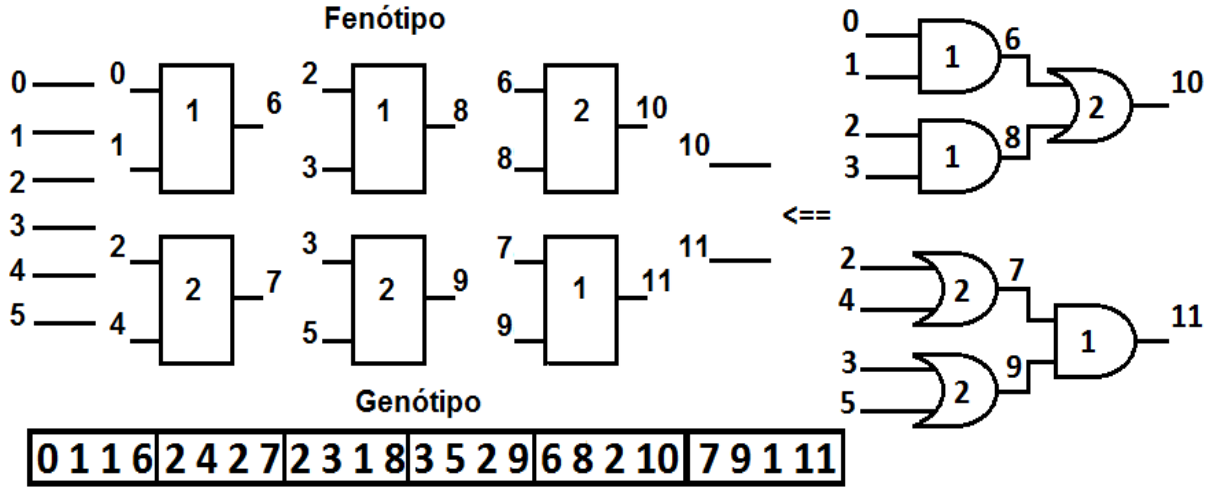

Figura 2.7 Exemplo de Programação Genética Cartesiana (Fenótipo e Genótipo).

Em PGC, os programas são representados sob a forma de grafo. Esses grafos são representados como uma rede bidimensional de nós computacionais. Os genes que constituem o genótipo em PGC são representados por números inteiros, onde um

nó recebe os respectivos dados, que executa as operações do nó sobre aqueles dados. Quando o genótipo é decodificado, alguns nós podem ser ignorados. Isto acontece quando as saídas de um nó não são utilizadas nos cálculos dos dados. Chama-se o programa que resulta da decodificação de um genótipo, de fenótipo. O genótipo de PGC tem um comprimento fixo. No entanto, o tamanho do fenótipo (em termos do número de nós computacionais) pode ser qualquer valor desde zero até o número de nós definidos no genótipo. O mapeamento genótipo-fenótipo usado em PGC é uma de suas características principais (MILLER, 2011).

2.3 A Ferramenta GPLAB para MATLAB

MATLAB é um ambiente interativo para computação numérica, visualização e programação, criada para o uso técnico científico, onde se podem analisar dados, desenvolver algoritmos e criar modelos e aplicações. A linguagem é bem flexível, com muitas ferramentas e funções matemáticas embutidas. O MATLAB permite explorar múltiplas abordagens e chegar a uma solução mais rapidamente, se comparado a outras linguagens de programação tradicionais, como C / C + + ou Java (MATHWORKS, 2014). Sua utilização traz o benefício de se ter pouco risco de erros de implementação, devido principalmente a necessidade de pouca codificação se comparada as outras linguagens mencionadas. O uso do MATLAB no meio acadêmico tem-se tornado frequente.

A GPLAB é um conjunto de ferramentas de programação genética para o MATLAB e é capaz de suportar as principais características da programação genética. Este trabalho aborda as principais funcionalidades e características desta ferramenta. Sua arquitetura é modular e parametrizada e pode ser facilmente usada e modificada.

Para executar a ferramenta sem parâmetros pré-definidos utilizam-se os seguintes comandos:

[vars, b] = gplab(g, n);

[vars, b] = gplab(g, vars);

Esta função inicializa todos os parâmetros do algoritmo com os valores padrão e o executa para ‘g’ gerações com ‘n’ indivíduos. O usuário é questionado sobre a localização dos arquivos de dados a usar. Ela retorna ‘vars’, que contém todas as

variáveis do algoritmo, e ‘b’ que representa o melhor indivíduo encontrado. A segunda função continua uma execução iniciada anteriormente por mais ‘g’ gerações, e também precisa de ‘vars’ como um argumento de entrada.

Para executar com parâmetros pré-definidos, deve-se definir a variável que vai conter os parâmetros com:

p = resetparams;

Então, ‘p’ irá receber a lista de parâmetros com valores *default*, e a partir desse momento, pode-se redefinir os parâmetros.

2.3.1 Funções e Terminais

GPLAB tem um grupo de funções predefinidas, que não é necessário selecionar para serem usadas, mas qualquer função do MATLAB pode ser usada como uma função ou terminal da Programação Genética. Os terminais podem ser qualquer constante ou uma função do MATLAB.

Para modificar o conjunto de funções é preciso executar a instrução abaixo:

p=setfunctions(p,’func1’,Aridade1,’func2’,Aridade2);

onde: ‘p’ é a variável que contém os parâmetros, ‘setfunctions’ é a instrução que muda o conjunto de funções, ‘func1’ é o nome da função que se deseja usar, ‘Aridade1’ é o número de argumentos da função ‘func1’, ‘func2’ é o nome de outra função que se deseja usar, ‘Aridade2’ é o número de argumentos da função ‘func2’.

Para modificar o conjunto de terminais é preciso executar a instrução abaixo:

p=setterminals(p,’term1’,’term2’);

onde: ‘p’ é a variável que contém os parâmetros, ‘setterminals’ é a instrução que muda o conjunto de terminais, ‘term1’ é o nome do terminal que se deseja usar, ‘term2’ é o nome do outro terminal que se deseja usar.

2.3.2 População Inicial

Ao iniciar uma nova execução da GPLAB é produzida uma população inicial de indivíduos, escolhendo aleatoriamente elementos dos conjuntos de funções e terminais. Os indivíduos criados têm tamanho controlado, por profundidade ou tamanho máximo; profundidade é a quantidade de níveis da árvore e tamanho é a quantidade total de nós da árvore. O parâmetro ‘inicmaxlevel’ indica o valor limite para

ambos os casos, o parâmetro ‘depthnodes’, com valor 1, seleciona limite por profundidade, e valor 2, seleciona limite por tamanho. GPLAB tem três métodos de construção de indivíduos;

- **Fullinit**: este método procura criar indivíduos com o tamanho máximo definido e com árvores equilibradas, ou seja, árvores cheias. Quando o limite por profundidade é usado, monta indivíduos até atingir o limite de níveis especificado em ‘inicmaxlevel’. Quando o limite por tamanho é usado, GPLAB monta indivíduos selecionando funções até se aproximar do tamanho limite e depois escolhe os terminais, de forma que não ultrapasse o limite de nós; porém, pode ter um número de nós menor que o especificado neste caso. Na Figura 2.8, tem-se um exemplo, onde se configurado como profundidade 3 ou tamanho 8, o resultado seria o mesmo, uma árvore com 3 níveis, no caso, a GPLAB não considera os terminais nessa contagem, e por tamanho 7 nós. Ou seja, no caso de profundidade 3 o indivíduo atinge o tamanho especificado de níveis, enquanto que no caso de tamanho 8 o indivíduo não consegue atingir o número de 8 nós, sendo possível apenas 7 para se obter uma árvore cheia.

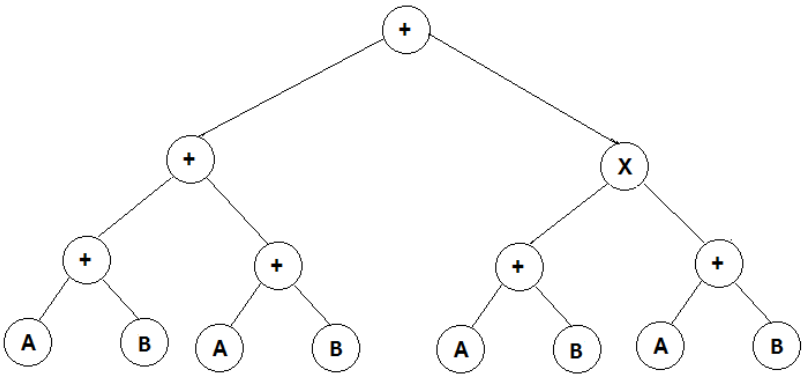

Figura 2.8 Exemplo de ‘Fullinit’.

- **Growinit:** neste método, o procedimento inicia escolhendo os terminais e depois as funções, crescendo a árvore aleatoriamente até o limite ‘inicmaxlevel’; este método cria árvores que podem ser muito desequilibradas, com ramos de diferentes tamanhos. Na Figura 2.9, tem-se um exemplo, onde se escolhido profundidade 3 ou tamanho 6, o resultado seria o mesmo, ou seja, a árvore com 3 níveis e com 6 nós. E

neste caso, o indivíduo consegue atingir os limites configurados, pois, a árvore pode ser desequilibrada.

- **Rampedinit:** este método cria metade dos indivíduos da população utilizando o processo ‘fullinit’ e a outra metade é criada utilizando-se o processo ‘growinit’.

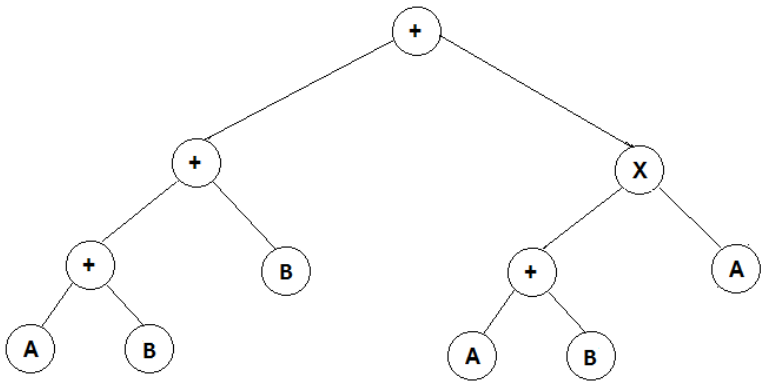

Figura 2.9 Exemplo de ‘Growinit.

A população é inicializada assim que a ferramenta GPLAB é executada, controlando a população inicial, de acordo com os seguintes parâmetros:

p.depthnodes='Valor';

p.inicmaxlevel=Inteiro;

p.initpoptype='PopTipo';

onde ‘p’ é a variável que contém os parâmetros usados para definir a população inicial; ‘depthnodes’ é o item da lista que contém o método do limite; ‘Valor’: se for ‘1’ seleciona o controle por profundidade e se for ‘2’ seleciona por tamanho; o parâmetro ‘inicmaxlevel’ é o item da lista que contém um valor inteiro que é o limite da quantidade de nós ou níveis dos indivíduos; ‘initpoptype’ é o item da lista que contém o método de criação dos indivíduos, sendo ‘PopTipo’ o nome do método que pode ser: ‘fullinit’, ‘growinit’ ou ‘rampedinit’.

2.3.3 Seleção

Redefine o método de seleção dos pais para serem submetidos aos operadores genéticos; são quatro os métodos de amostragem dos pais:

- **Roulette:** método da roleta, onde cada indivíduo recebe uma fatia da roleta correspondente à sua aptidão; ao girar a roleta um indivíduo é

selecionado aleatoriamente; indivíduos com melhor aptidão terão mais chances de serem escolhidos.

- **Sus:** método da roleta, no qual cada indivíduo tem a mesma chance; não considera a aptidão.
- **Tournament:** método de torneio que seleciona um número aleatório de indivíduos e separa o de melhor aptidão.
- **Lexictour:** como em *tournament,* seleciona um grupo de indivíduos e separa o melhor, porém, se dois indivíduos tem a mesma aptidão, aquele com menor número de nós é escolhido.
- **Doubletour:** ou duplo torneio, onde o primeiro é um torneio por aptidão e o segundo é pelo tamanho dos vencedores do primeiro.

Para modificar o modo de seleção é preciso executar a instrução abaixo;

p.sampling = ’método’;

onde ‘p’ é a variável que contém os parâmetros, ‘sampling’ é o item da lista que contém o método de seleção e ‘método’ é o nome do método que se deseja utilizar.

2.3.4 Operadores

A ferramenta GPLAB possui um conjunto de operadores genéticos previamente disponíveis, e outros podem ser criados através de funções do MATLAB; estes operadores genéticos são:

- **Crossover:** é a recombinação genética sexuada, onde nela são escolhidos dois ou mais pais que trocam material entre si, gerando dois ou mais filhos.
- **Mutation:** é a mutação; um nó é escolhido do indivíduo original e substituído por uma sub-árvore aleatória, criando-se outro indivíduo.
- **Shrink Mutation:** é uma mutação em que uma sub-árvore do indivíduo original é escolhida e substituída por uma parte desta sub-árvore, ou seja, de uma sub-árvore desta sub-árvore.
- **Swap Mutation:** é uma mutação em que duas sub-árvores do indivíduo original são escolhidas e trocadas entre si.

Para utilizar esses operadores é preciso executar a instrução abaixo:

p = setoperators (p, ’operador’, pais, filhos);

onde: ‘p’ é a variável que contém os parâmetros; ‘setoperators’ é a instrução que muda os operadores; ‘operador’ é o operador que se deseja usar; ‘pais’ representa o número de pais que são recombinados e ‘filhos’ o número de filhos que se deseja obter.

2.3.5 Aptidão

GPLAB tem um método padrão para o cálculo de regressão simbólica e paridade, o ‘regfitness’, que é o padrão ao usar a ferramenta. Existem mais dois métodos disponíveis com a ferramenta para o problema de formiga artificial. Porém, uma nova função do MATLAB pode ser elaborada, de forma a calcular adequadamente a aptidão para o problema proposto. Os métodos disponíveis na ferramenta são:

- **Regfitness:** calcula a aptidão pelo custo, ou seja, a diferença absoluta entre o valor esperado e o valor encontrado ao executar o indivíduo; os mais aptos são aqueles com aptidão menor. O parâmetro ‘lowerisbetter’ deve ser 1, obrigatoriamente.
- **Antfitness:** soma os locais com comida por onde a formiga passou em determinado número de passos; indivíduos com aptidões maiores são melhores. O parâmetro ‘lowerisbetter’ deve ser 0 obrigatoriamente.
- **Antfitness lib:** encontra o número de locais com comida que a formiga não encontrou em determinado número de passos; indivíduo com aptidão menor é melhor. E o parâmetro ‘lowerisbetter’ deve ser 1, obrigatoriamente.

A aptidão é redefinida com:

p.calcfitness = ‘FuncFitness’;

onde ‘p’ é a variável que contém os parâmetros, ‘calcfitness’ é o item da lista que contém o nome da função de aptidão, ‘FuncFitness’ é o nome da função que calcula a aptidão e que fornece um valor de aptidão para a ferramenta GPLAB,

p.lowerisbetter = Valor;

onde ‘p’ é a variável que contém os parâmetros, ‘lowerisbetter’ é o item da lista que contém o valor que seleciona a forma como a aptidão é avaliada pela ferramenta GPLAB, e ‘Valor’ é um número inteiro; quando ‘0’, a ferramenta considera o valor de

aptidão maior, como melhor, caso contrário considera o valor de aptidão menor, como melhor.

2.3.6 Arquivos de Entrada e Saída

A ferramenta utiliza alguns arquivos em formato de texto (.txt) para a sua execução, onde dois desses arquivos são muito importantes e obrigatórios para seu funcionamento adequado, pois armazenam os casos de aptidão desejados; um deles contém os dados das entradas (datafilex) e o outro contém os resultados desejados (datafiley), para os casos de regressão simbólica e paridade. Para os casos de *artificial ant*, o primeiro contém a trilha de alimentos, no formato de uma matriz binária, e o segundo contém o número de alimentos.

Os arquivos são especificados com;

p.datafilex = ‘NomeEntrada’;

p.datafiley = ‘NomeSaida’;

onde ‘p’ é a variável que contém os parâmetros e ‘datafilex’ é o item da lista que contém o nome do arquivo de entrada; ‘NomeEntrada’ é o nome do arquivo de entrada, ‘datafiley’ é o item da lista que contém o nome do arquivo de resultados, e ‘NomeSaida’ é o nome do arquivo de resultados.

Os dados desses arquivos do tipo texto necessitam ser importados para a ferramenta em variáveis do algoritmo para que possam ser usadas no processo de evolução. Isso é feito conforme é determinado pelo processo especificado em ‘files2data’ e há dois métodos diferentes disponíveis na ferramenta; outros podem ser elaborados para atender melhor a evolução que se deseja. São eles:

- ‘xy2inout’: usado em problemas de regressão e paridade, que é padrão.
- ‘anttrail’: usado em problemas de formigas artificiais.

Para redefinir ‘files2data’:

p.files2data = ‘NomeImportação’;

onde ‘p’ é a variável que contém os parâmetros, ‘files2data’ é o item da lista que contém o nome da função de importação e ‘NomeImportação’ é o nome da função utilizada.

2.3.7 Sobrevivência

Após a ferramenta efetuar as operações genéticas e produzir novos indivíduos para uma nova população, é executado um procedimento de sobrevivência para escolher um número determinado de indivíduos para essa nova população, esse número é determinado conforme o parâmetro de sobrevivência, e isso se dá em duas etapas. Na primeira etapa todos os indivíduos, tanto pais, quanto filhos, são enumerados por um método de elitismo. Na segunda etapa, a sobrevivência é concedida para cada indivíduo enumerado da lista, conforme o parâmetro de sobrevivência.

Os parâmetros de elitismo são:

- **Replace:** todos os filhos são ordenados em primeiro lugar e recebem a sobrevivência antes dos pais, que são ordenados e recebem a sobrevivência depois, assim, os filhos obtidos substituem seus pais.
- **Keepbest:** o melhor indivíduo do conjunto de pais e filhos é mantido e receberá sua sobrevivência. Os demais indivíduos são processados como em ‘replace’.
- **Halfelitism:** metade dos que recebem a sobrevivência são os melhores filhos, e a metade restante são dos melhores pais. Sendo a população final composta de filhos e pais na mesma proporção.
- **Totalelitism:** todos os indivíduos são ordenados por aptidão, tanto pais e filhos, e vão receber a sobrevivência apenas os melhores até o valor limite da população.

Para alterar o parâmetro de elitismo é preciso executar a instrução abaixo:

p.elitism = ‘método’;

onde ‘p’ é a variável que contém os parâmetros, ‘elitism’ é o item da lista que contém o método; ‘método’ é o nome do método escolhido.

Os parâmetros de sobrevivência são:

- **Fixedpopsize:** o tamanho é fixo, e o número de indivíduos na população permanece o mesmo; os indivíduos mais aptos sobrevivem até atingir o número estipulado (n) na ocasião da execução da GPLAB.

- **Resources:** o número de indivíduos pode variar em função de várias opções, onde esta técnica é experimental e não é abordada em mais detalhes.
- **Pivotfixe:** o número de indivíduos pode variar em função da economia de recursos e esforço computacional, sendo esta técnica experimental e não é abordada em mais detalhes.

Para alterar o parâmetro de sobrevivência é preciso executar a instrução abaixo:

p.survival = ‘método’;

onde ‘p’ é a variável que contém os parâmetros, ‘survival’ é o item da lista que contém o método, e ‘método’ é o nome do método escolhido.

2.3.8 Ferramenta Aplicada a um Caso de Regressão

A ferramenta é aplicada a um caso de regressão para a função XOR, conforme os casos de aptidão descritos na Tabela 2.3.

Tabela 2.1–Casos de Aptidão.

Arquivo datafilex		**Arquivo datafiley**
Entrada X1	**Entrada X2**	**Saída**
0	0	0
0	1	1
1	0	1
1	1	0

A função para executar a ferramenta GPLAB é descrita na Figura 2.10.

A função é criada com:

```
function [v,b]= evolucaoXOR
```

Os parâmetros iniciais necessitam ser carregados em uma lista 'p', com:

```
p=resetparams;
```

O operador de cruzamento é utilizado com dois pais gerando dois filhos, com:

```
p=setoperators(p,'crossover',2,2);
```

As funções lógicas (‘nand, ‘nor’, ‘and e ‘or) necessárias à solução são carregadas com:

p=setfunctions(p,'nand',2,'nor',2,'and',2,'or',2);

Os terminais utilizados são os definidos pelos arquivos de aptidão e definido como segue:

p=setterminals(p);

Os arquivos com os casos de aptidão são:

p.datafilex='Textx.txt';

p.datafiley='Texty.txt';

```
function [v,b]=evolucaoXOR
%evolucaoXOR demonstração do GPLAB para um circuito XOR.
%
fprintf('Running XOR demo...');
p=resetparams;
p=setoperators(p,'crossover',2,2);
p=setfunctions(p,'nand',2,'nor',2,'and',2,'or',2);
p=setterminals(p);
p.datafilex='Textx.txt';
p.datafiley='Texty.txt';
p.depthnodes='1';
[v,b]=gplab(20,50,p);
drawtree(v.state.bestsofar.tree);
```

Figura 2.10 Função para caso de regressão de XOR.

O controle de tamanho dos indivíduos é definido como profundidade da árvore, nesse caso, como ‘inicmaxlevel’ não é estipulado a o valor para este é definido por padrão da ferramenta em até 6 níveis:

p.depthnodes='1';

A ferramenta é executada com população de 50 indivíduos e por 20 gerações, conforme segue:

[v,b]=gplab(20,50,p);

O melhor indivíduo encontrado é mostrado com o comando abaixo:

drawtree(v.state.bestsofar.tree);

O resultado da execução dessa aplicação é mostrado na Figura 2.11, e equivale à expressão ‘*nor(and(X2,X1),nor(X2,X1))*’.

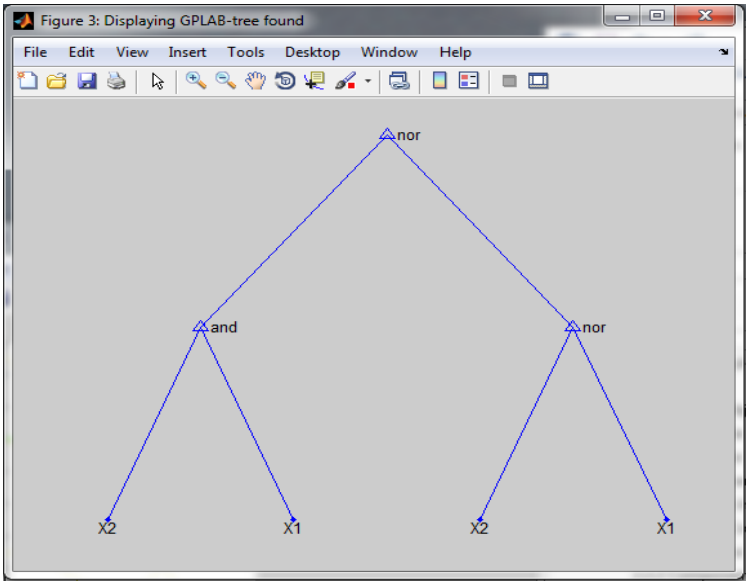

Figura 2.11 Resultado encontrado.

2.4 Processamento Digital de Imagens

O processamento digital de imagens consiste em manipular as imagens adquiridas de um *hardware* de aquisição, de forma que essas apresentem condições para uma avaliação adequada por um especialista; o sistema de processamento de imagens é mostrado na Figura 2.12 de forma simplificada. Existem diversas técnicas possíveis de serem aplicadas para tal processamento e uma delas ocorre por meio de técnicas de filtragem.

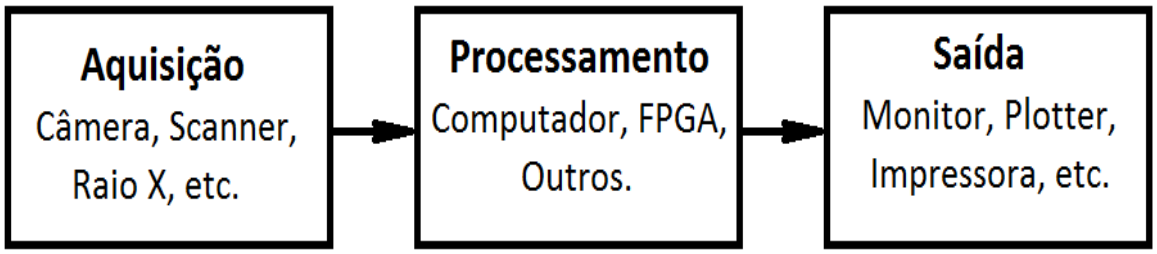

Figura 2.12 Sistema de processamento de Imagens.
Adaptado de (MARQUES e VIEIRA, 1999).

2.4.1 Técnicas de filtragem de imagens

A filtragem de uma imagem tem como objetivo obter uma imagem de saída mais adequada ao uso que se pretende, portanto, os resultados desejados dependem de cada problema na qual a filtragem é aplicada.

As muitas técnicas de filtragem são divididas em: técnicas no domínio espacial e técnicas no domínio da frequência (GONZALEZ e WOODS, 2001).

2.4.1.1 Filtragem no domínio espacial

A filtragem no domínio espacial é uma técnica que atua diretamente na matriz de *pixels* da imagem digitalizada, executando operações lógicas e aritméticas sobre um conjunto desses *pixels*. Esta é caracterizada matematicamente como:

$$g(x,y) = T[f(x,y)] \quad (1)$$

onde *g(x,y)* é a imagem após o processamento, *f(x,y)* é a imagem original e *T* é um operador em *f*, definido em uma certa vizinhança de *(x,y)*.

Essa vizinhança ao redor de *(x,y)*, ou seja, do *pixel* que está sendo tratado, em geral é do tipo 8-vizinhança, e descreve uma matriz 3 x 3 em que este *pixel* está sob o centro desta matriz. Para o processamento da filtragem, essa janela denotada pela matriz é movida *pixel* a *pixel* e é aplicado o operador *T* para encontrar o valor de *g* naquele ponto.

2.4.1.2 Filtragem no domínio da frequência

Filtragem no domínio da frequência é a técnica que utiliza o Teorema da Convolução como base matemática para tratar as imagens, como demonstra a equação, a seguir, no domínio espacial:

$$g(x,y) = f(x,y) * h(x,y) \quad (2)$$

onde *g(x,y)* é a imagem resultante, *f(x, y)* é a imagem original e *h(x,y)* é um operador linear.

Pelo Teorema da Convolução pode-se obter, a equação no domínio da frequência:

$$G(u,v) = F(u,v)H(u,v) \quad (3)$$

onde *G*, *F* e *H* são as transformadas de Fourier de *g*, *f* e *h*, respectivamente.

2.4.2 Morfologia Matemática

Morfologia Matemática é o estudo da estrutura geométrica das entidades presentes em uma imagem, sendo um processo de filtragem no domínio espacial; suas ferramentas apresentam uma maior facilidade de aplicação em sistemas embarcados. Esta é usada para extrair componentes de imagem, de forma que sejam úteis na representação e descrição de regiões, tais como a forma, limites, esqueletos e o casco convexo, e pode ser usada também para o processamento de imagens, tais como filtragem, afinamento e poda. As operações básicas em morfologia matemática são a dilatação, erosão, abertura e fechamento (GONZALEZ e WOODS, 2001).

2.4.2.1 Dilatação

Sendo *A* e *B* conjuntos no espaço e *Ø* um conjunto vazio, a dilatação de *A* por *B*, é definida como:

$$A \oplus B = \left\{ x \middle| \left(\hat{B}\right)_x \cap A \neq \emptyset \right\} \tag{4}$$

Assim, a dilatação de *A* por *B* é o conjunto de todos os deslocamentos (x), tais que ocorre uma interseção não vazia de $\left(\hat{B}\right)_x$ e *A*. Com base nesta interpretação, a equação anterior pode ser reescrita como:

$$A \oplus B = \{x | [\left(\hat{B}\right)_x \cap A] \subseteq A\} \tag{5}$$

O conjunto *B* é denominado elemento estruturante. Tal processo pode ser visualizado como sendo uma expansão de uma figura contida em uma imagem.

2.4.2.1 Erosão

Sejam *A* e *B* conjuntos no espaço. A erosão de *A* por *B*, é definida conforme a Equação (6). Em outras palavras, significa dizer que a erosão de *A* por *B* resulta no conjunto de deslocamentos (*x),* tais que, *B*, transladado de *x*, esteja totalmente contido em A. Pode-se simplificar essa descrição como sendo um processo de encolhimento de uma figura contida em uma imagem.

$$A \ominus B = \left\{ x \middle| \left(\hat{B}\right)_x \subseteq A \right\} \tag{6}$$

2.4.2.1 Abertura

A abertura de um conjunto *A* por um elemento estruturante *B*, é definida como:

$$A \circ B = (A \ominus B) \oplus B \qquad (7)$$

Portanto, a abertura de *A* por *B* é simplesmente a erosão de *A* por *B* seguida de uma dilatação do resultado por *B*. A Abertura suaviza contornos e elimina proeminências.

2.4.2.1 Fechamento

O fechamento do conjunto *A* pelo elemento estruturante *B*, é definido como:

$$A \cdot B = (A \oplus B) \ominus B \qquad (8)$$

Então, o fechamento é a dilatação de *A* por *B* seguida da erosão do resultado pelo mesmo elemento estruturante *B*. O fechamento produz uma fusão de pequenas quebras, elimina pequenos orifícios, alarga golfos estreitos e preenche vãos.

2.4.2.1 Outros operadores

Em filtragem no domínio espacial é possível aplicar outros operadores, que combinam *pixels* vizinhos com o processado, dentro de uma janela previamente definida. Supondo-se uma janela de 3×3 *pixels*, qualquer dos 9 *pixels* podem ser combinados para se obter um *pixel* de saída, como apresentado na Figura 2.13.

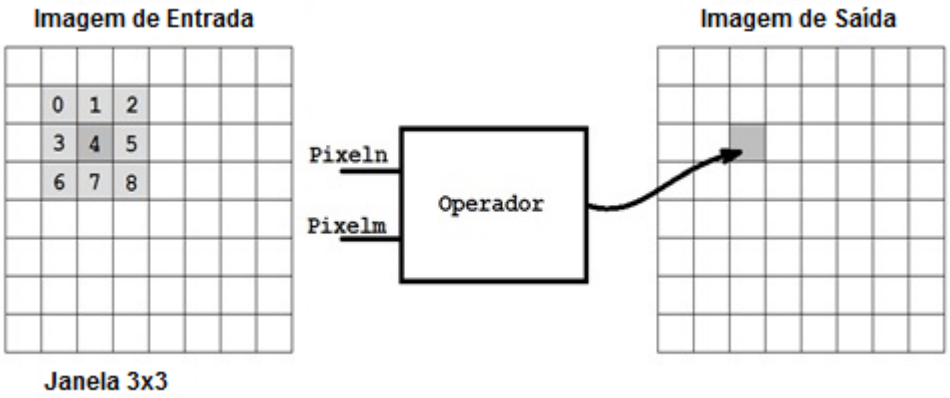

Figura 2.13 Processamento com outros operadores.
Adaptada (MARQUES e VIEIRA, 1999).

2.5 Conceitos de Sistemas Embarcados

Sistema Embarcado é um sistema microprocessado, que une *hardware* e *software*, e que está totalmente dedicado e incorporado a um dispositivo ou sistema que controla. Tais sistemas se tornaram possíveis devido ao avanço da capacidade dos circuitos integrados (CI), que muitas vezes agregam *hardware* e *software* em um mesmo circuito integrado e o crescente aumento na qualidade dos compiladores. Estão presentes diariamente em nossas vidas, desde o automóvel ao micro-ondas (VAHID e GIVARGIS, 1999).

Sistemas embarcados possuem como características:

- Única Função: o sistema embarcado executa apenas um programa.
- Customização: o sistema embarcado é altamente customizado, compacto, eficiente e eficaz.
- Tempo Real: os sistemas embarcados reagem muito rapidamente às alterações de seus estímulos, trabalhando no que se chama de tempo real, ou seja, respondem relativamente rápido.

Na Figura 2.14 é apresentado o processo de operação de um sistema embarcado, onde se têm:

- Aquisição: processo de aquisição dos sinais externos para o dispositivo. Estes sinais são necessários para a execução das tarefas desejadas.
- Processamento de sinais: processo de tratamento dos sinais adquiridos. Este processo manipula os sinais de forma que possam ser usados na execução das rotinas.
- Rotinas embarcadas: execução das tarefas programadas; neste processo, o sistema efetua as tarefas para as quais foi criado.
- Controle: processo de controle dos sinais de saída do dispositivo e que comandam dispositivos do sistema.

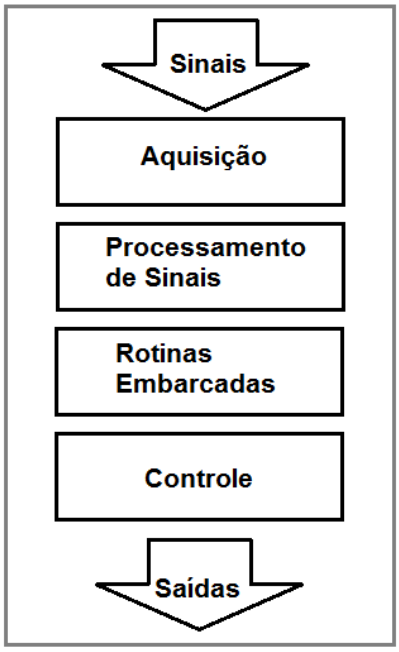

Figura 2.14 Processo simplificado de operação de um sistema embarcado.

2.5.1 Tecnologias Empregadas em Sistemas Embarcados

Sistemas embarcados utilizam circuitos integrados devido à necessidade de serem extremamente customizados e compactos, e diferentes tecnologias de fabricação desses circuitos integrados (CI) são utilizadas em sistemas embarcados. Essas tecnologias são:

2.5.1.1 Totalmente customizado ou VLSI

Em VLSI (*Very Large Scale Integration*), o circuito integrado (CI) é totalmente customizado, ou seja, é feito um projeto especifico de circuito para ser confeccionado pelo fabricante de CIs; o tempo e o custo para iniciar a produção é alto, porém, o desempenho é extremamente alto, por apresentar alta densidade, e baixo custo de produção devido a um grande volume de produção.

2.5.1.2 Microcontroladores

Microcontroladores são CIs do tipo VLSI com características de um microprocessador, ou seja, são capazes de executar um programa, porém para um objetivo especifico de um sistema embarcado, e possuem caraterísticas adicionais; além de possuírem componentes lógicos e aritméticos, possuem armazenamento de

dados, armazenamento de programa, conversores analógicos e dispositivos periféricos.

2.5.1.3 Parcialmente Customizado ou ASIC

Em ASIC (*Application-Specific IC*), o circuito integrado (CI) é parcialmente customizado; um projeto especifico do circuito é feito, porém, muitas partes são parcialmente construídas, deixando um grupo de portas em um conjunto de matrizes que podem ser configuradas posteriormente, adicionando-se as funcionalidades não previstas.

2.5.1.4 PLD

Um PLD (*Programmable Logic Device*) consiste em um circuito integrado (CI), onde o circuito é construído, em sua totalidade, com portas lógicas que podem ser conectadas entre si dentro do CI por meio de uma matriz de configuração, que são conectadas posteriormente conforme a necessidade do fabricante do sistema embarcado. Podem não oferecer as mesmas qualidades do VLSI ou ASIC, mas seu custo inicial de produção é muito mais baixo, pois é possível utilizar CIs já existentes no mercado e produzidos em larga escala.

2.5.1.5 FPGA

FPGA (*Field Programmable Gate Array*) é um tipo mais complexo de PLD, que o uso vem crescendo e ganhando popularidade, por oferecer uma maior conectividade entre os blocos de lógica, portanto, pode implementar projetos de maior complexidade. Sua arquitetura é composta por três blocos básicos:

- BLC (Blocos Lógicos de Configuração) são conjuntos de circuitos idênticos, formados pela reunião de *flip-flops* e lógica combinacional numa implementação do tipo LUT (*Look-Up Table*), capaz de implementar qualquer tabela verdade.
- BES (Bloco de Entrada e Saída) são circuitos que fazem a conexão das entradas e saídas com os blocos lógicos internos.
- Matriz de Conexão: matriz utilizada para fazer a conexão de forma programável entre os blocos.

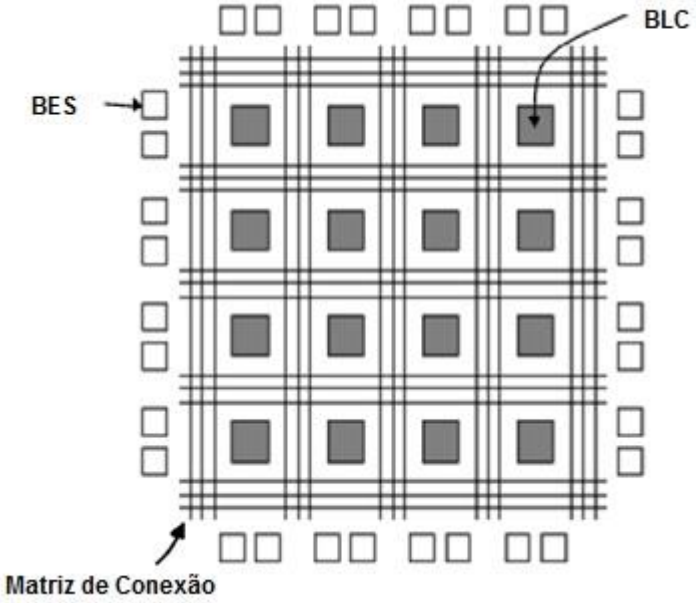

Figura 2.15 Estrutura Simplificada FPGA, adaptado (ALTERA CORPORATION, 2007).

A programação da FPGA ocorre em nível de *hardware*, como demonstra a Figura 2.15 e a Figura 2.16; a Matriz de Conexão é capaz de realizar a ligação entre diversos pontos da estrutura, e cada ponto de ligação é um *bit* de memória da FPGA.

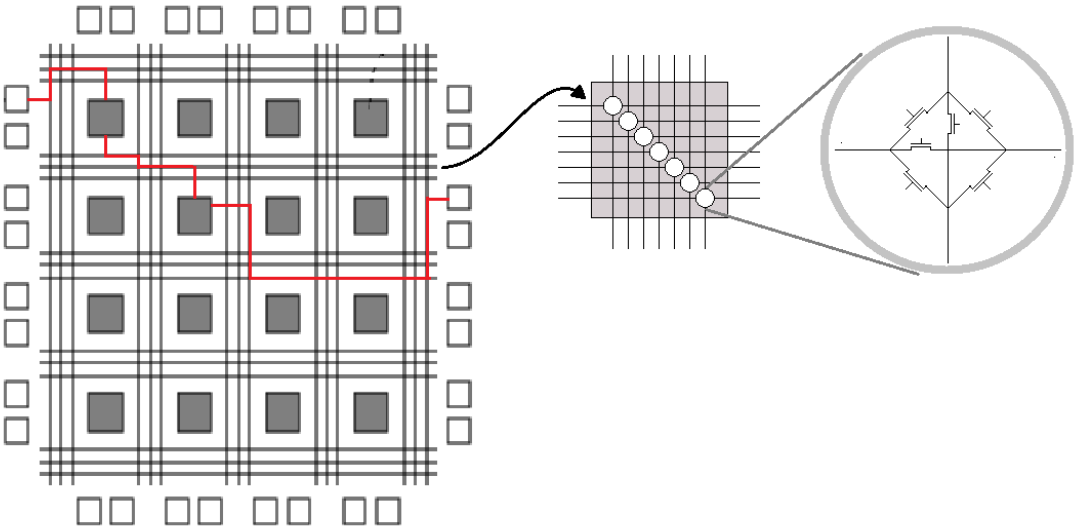

Figura 2.16 Estrutura da *Switch Matrix,* adaptado (ALTERA CORPORATION, 2007).

A programação modifica as ligações da Matriz de Conexão, em que cada ponto de ligação é um *bit* de memória, realizando assim as conexões entre as entradas e saídas das LUTs (*Look-Up Table*) das CLBs (*Configuration Logical Blocks*) e os IOBs (*Input/Output Block*), como mostrado nas Figuras 2.16 e 2.17.

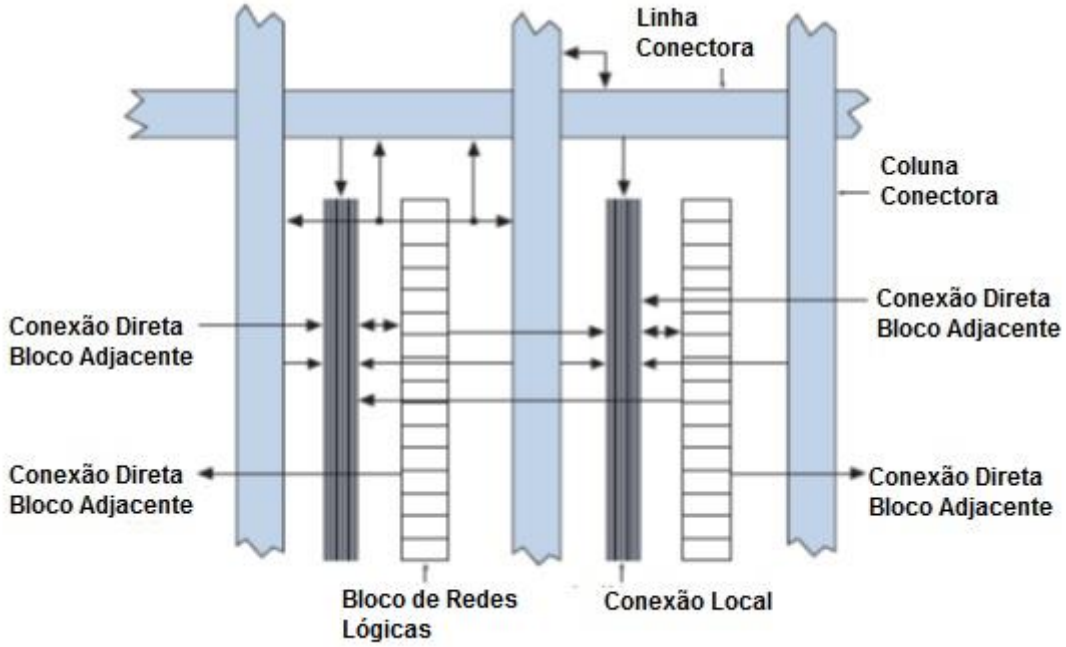

Figura 2.17 Estrutura de uma FPGA da Família Cyclone II da Altera, adaptado (ALTERA CORPORATION, 2007).

Na Figura 2.18 é possível comparar as tecnologias descritas, quanto ao seu custo e tempo de desenvolvimento, contra suas velocidades, densidades, complexidades e volume de produção.

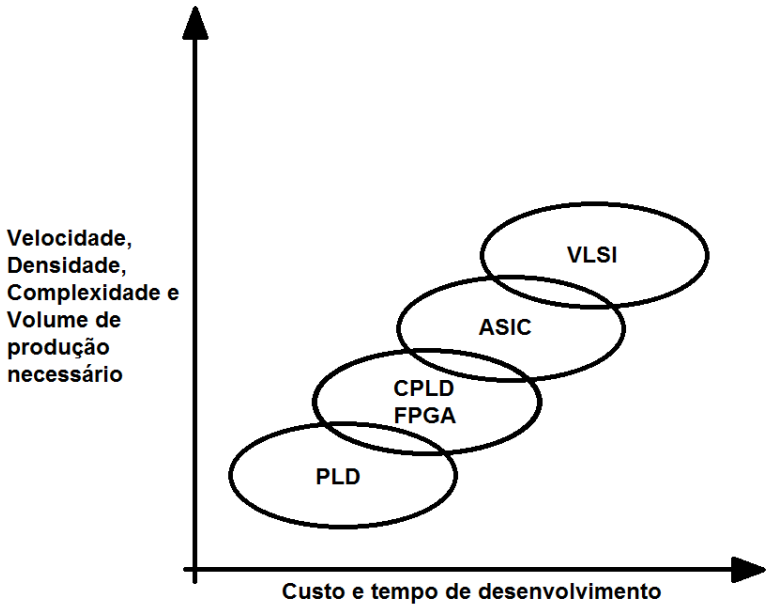

Figura 2.18 Comparação entre as arquiteturas (GOEL, 2016).

2.6 *Hardware* evolutivo em FPGA

Hardware evolutivo é uma técnica que emprega os conceitos de sistemas evolutivos para encontrar um *hardware* adequado a um problema que se deseja resolver.

Duas técnicas de HE se destacam: HE extrínseca, na qual a evolução do *hardware* é feita em algum dispositivo dedicado, geralmente em computadores, e um *hardware* pode ser construído; e a HE intrínseca, que utiliza dispositivos lógicos programáveis, tal como a FPGA, permitindo a evolução e a reconfiguração do *hardware*.

Sendo a FPGA um dispositivo de alto desempenho e grande flexibilidade de reconfiguração, esta tem sido utilizada amplamente como plataforma de HE.

2.6.1 HE extrínseca em FPGA usando *bitstream*

Para essa técnica, cada cromossomo gerado por uma evolução é configurado na FPGA por meio de um *software* proprietário que realiza a configuração por uma porta de comunicação entre o computador e a FPGA. Essa abordagem tem limitações: granularidade extremamente fina para evoluir circuitos mais complexos, comprimento de cromossomo extremamente longo e grande tempo de reconfiguração (MILLER, 1999) (MILLER, 2011).

2.6.2 Arquitetura Virtual Reconfigurável (AVR)

AVR é uma técnica de HE intrínseca que implementa uma segunda camada de configuração no dispositivo FPGA, e consiste em elaborar um elemento com funções configuráveis, ou seja, Elemento de Função (EF), e replicar esse elemento em uma matriz de $n \times m$ elementos, sendo possível reconfigurar as conexões entre estas e as funções que podem assumir.

Nessa técnica o *bitstream* não é manipulado para reconfigurar a FPGA. A proposta é utilizar a arquitetura como uma representação de alto nível para funções do circuito desejado, reduzindo a complexidade do processo de reconfiguração, enquanto a velocidade do processo é aumentada. Com essa segunda camada de

configuração, o *bitstream* precisa ser enviado apenas uma vez para configurar a FPGA e não muda durante a evolução. A evolução do circuito é feita pela própria FPGA.

Durante a evolução do circuito, um cromossomo, bem menor que na abordagem anterior, é utilizado para reconfigurar a matriz de EFs.

Essa técnica também tem suas limitações: uso de grande parte da FPGA para evoluir o circuito, utiliza grandes quantidades de memória, ou seja, tem um custo alto de *hardware* e a necessidade de aplicar os testes de aptidão diretamente no circuito, dificultando que esse seja utilizado em testes práticos.

2.7 Considerações finais

Neste capítulo foi abordado que a programação genética é uma poderosa ferramenta na solução de problemas de programação, onde a solução é de difícil dedução, e como se pode utilizar seus conceitos para solucionar tais problemas. Foi visto que sistemas embarcados são uma realidade de nossos dias em que a necessidade de sistemas cada vez mais eficientes é uma constante. O uso da tecnologia de FPGA em sistemas reconfiguráveis constitui uma poderosa ferramenta na elaboração de sistemas embarcados de alto desempenho e de baixo consumo energético.

Capítulo 3

REVISÃO RELACIONADA

Neste capitulo é realizada uma revisão relacionada a hardware evolutivo empregado em FPGA, iniciando com uma comparação entre os diversos trabalhos da área e em seguida descrevendo os de maior interesse para o contexto atual.

3.1 Considerações iniciais

O principal objeto de estudo deste trabalho de mestrado é EHW, empregado em FPGA, para solução de problemas do mundo real; assim os seguintes problemas são abordados, neste texto: a geração de soluções de circuitos lógicos e a geração de filtros de imagem. Portanto, neste capítulo, os diversos trabalhos encontrados utilizando EHW, empregado em FPGA, são comparados com a intenção de se obter uma visão geral de sua evolução e seu atual estado da arte; assim dois problemas são abordados de forma sucinta no presente capítulo, por serem de maior relevância para o trabalho corrente, pois oferecem dados adequados para fins de comparação.

Muitos trabalhos têm abordado o uso de *hardware* evolutivo empregando FPGA nos últimos anos. Neste trabalho, é feita uma introdução da evolução e uma comparação entre esses trabalhos, para se obter uma visão geral deste campo de pesquisa; sendo assim, o conjunto formado por esses artigos serviram como base para o desenvolvimento do presente trabalho de mestrado.

Os primeiros trabalhos utilizando EHW com FPGA ocorreram na década de 1990, utilizando AG para elaborar circuitos discriminadores de frequência, osciladores e filtros de imagem. Tais trabalhos utilizavam o chip Xilinx XC6200, um dispositivo FPGA configurado por *bitstream* de formato aberto, que facilitava a reconfiguração por

computadores externos. Porém, a família de dispositivos XC6200 foi descontinuada pela Xilinx em 1998, alegando que o formato de *bitstream* aberto estava permitindo a engenharia reversa do produto, e em seguida apresentou a família Virtex para substituir a primeira.

Devido aos limites impostos pela família Xilinx Virtex novas propostas de abordagens de EHW em FPGA foram apresentadas nas décadas de 2000 e 2010, onde essas propostas podem ser divididas em duas categorias: Arquitetura Virtual Reconfiguravel (VRA) e manipulação de bitstream direto. Esses trabalhos passaram a usar PG, PGC, estratégia evolutiva e programação celular além do AG, e foram usados para elaborar circuitos osciladores, circuitos tolerantes à falhas, somadores, multiplicadores, *benchmarks*, filtros de imagens, contadores e geradores de paridade (CANCARE, BHANDARI, *et al.*, 2011).

Estratégia evolutiva e programação celular não são abordados neste trabalho, mas, de forma sucinta, têm-se:

- Estratégia evolutiva como uma variação de AG que realiza uma adaptação dos parâmetros da evolução durante sua execução.
- Programação celular como um algoritmo que considera uma grade de células, com dimensões finitas e que podem assumir um número finito de estados. Ao executar o algoritmo, estados iniciais são inicializados aleatoriamente em toda a grade; para M iterações, e repeti-lo para um número de diferentes estados iniciais.

Uma rápida comparação entre os trabalhos encontrados é feita nas Tabelas 3.1 e 3.2.

Tabela 3.1 – Comparação dos trabalhos, adaptado (CANCARE, BHANDARI, *et al.*, 2011).

Autor	**Método**	**Tecnologia**	**Algoritmo**	**Caso Estudado**
(THOMPSON, 1996)	Evolução intrínseca	Xilinx XC6200	AG	Discriminador de frequência
(THOMPSON, 1998)	Evolução intrínseca	Xilinx XC6200	AG	Discriminador de frequência
(PORTER e BERGMANN, 1999)	Evolução intrínseca	Xilinx XC6200	AG	Filtro de imagem

(HUELSBERGEN e RIETMAN, 1999)	Evolução intrínseca	Xilinx XC6200	AG	Osciladores
(LEVI e GUCCIONE, 1999)	Evolução intrínseca	Xilinx XC4000	AG	Contadores de divisores de frequência
(TORRESEN, 1999)	Evolução intrínseca	Xilinx XC6200	AG	Reconhecimento de caracteres
(KALGANOVA, 2000)	Evolução extrínseca	N.A.	(1+λ) estratégia evolutiva	*Benchmark*
(TYRRELL, HOLLINGWORTH e SMITH, 2001)	Evolução extrínseca	Xilinx Virtex	(1+λ) estratégia evolutiva	Oscilador tolerante à falha
(GALLAGHER e VIAGRAHAM, 2002)	Evolução intrínseca	N.A.	AG compacto	Redes Neurais
(YASUNAGA, YOSHIHARA e KIM, 2003)	Evolução extrínseca	Xilinx Virtex	AG	Reconhecimento de gene humano
(SEKANINA e FRIEDL, 2004)	Evolução intrínseca	Xilinx Virtex II	PGC	Somadores, multiplicadores e geradores de paridade
(UPEGUI e SANCHEZ, 2005)	Evolução intrínseca	N.A.	AG	Nenhum
(UPEGUI e SANCHEZ, 2006)	Evolução intrínseca	Xilinx Virtex II	Programação Celular	Sincronização de mecanismos
(GLETTE, TORRESEN e YASUNAGA, 2006)	Evolução intrínseca	Xilinx Virtex II Pro	AG	Reconhecimento de imagem

(STOMEO, KALGANOVA e LAMBERT, 2006)	Evolução extrínseca	N.A.	(1+λ) estratégia evolutiva	Multiplicadores, geradores de paridade e MCNC *benchmarks*
(SEKANINA, 2007)	Evolução extrínseca	Simulador FPGA	PGC	Tolerância à falha
(VASICEK e SEKANINA, 2007)	Evolução intrínseca	Xilinx Virtex II Pro	AG e outros	Processamento de imagem
(WANG, PIAO e LEE, 2007)	Evolução intrínseca	Xilinx Virtex II	(1+λ) estratégia evolutiva	Somadores e multiplicadores
(OREIFEJ, AL-HADDAD, *et al.*, 2007)	Evolução intrínseca	Xilinx Virtex II Pro	AG	Somadores com autocorreção
(LAMBERT, KALGANOVA e STOMEO, 2009)	Evolução intrínseca	N.A.	Não descrito	Nenhum
(GLETTE, TORRESEN e HOVIN, 2009)	Evolução intrínseca	Xilinx Virtex II Pro	AG	Sonar e reconhecimento de imagem
(VASICEK, SEKANINA e BIDLO, 2010)	Evolução extrínseca	Xilinx Virtex II	PGC	Filtro de imagem
(CANCARE, SANTAMBROGIO e SCIUTO, 2010)	Evolução intrínseca	Xilinx Virtex IV	PGC	Gerador de paridade e contadores

Tabela 3.2 – Comparação dos trabalhos complemento.

Autor	**Método**	**Tecnologia**	**Algoritmo**	**Caso Estudado**
(WANG, CHEN e LEE, 2007)	Evolução intrínseca	Xilinx Virtex	(1+λ) estratégia evolutiva	Reconhecimento de caracteres e filtro de imagem
(REN, ZHAO, *et al.*, 2011)	Evolução intrínseca	Altera EP2C20F 484C7 Simulado	AG	Nenhum
(BARTOLINI, CANCARE, *et al.*, 2011)	Evolução intrínseca	Xilinx Virtex IV	AG	Nenhum
(JIANAN, ZHIWU, *et al.*, 2012)	Evolução intrínseca	N.A.	AG	Multiplicadores
(CANCARE, BARTOLINI, *et al.*, 2012)	Evolução intrínseca	Xilinx Virtex IV	AG	Gerador de paridade, comparador, somador e classificador
(WANG, WANG e LAI, 2013)	Evolução intrínseca	Xilinx Virtex V	AG	Nenhum
(SRIVASTAVA, GUPTA, *et al.*, 2014)	Evolução intrínseca	Xilinx Virtex II	PG	Detecção de falhas, somador e circuitos digitais
(GLETTE e KAUFMANN, 2014)	Evolução intrínseca	Xilinx Virtex V	AG	Nenhum

Dois trabalhos são abordados aqui por serem usados para fins de comparação, devido a sua relevância:

1- Projeto e simulação de circuito virtual reconfiguravel para um sistema tolerante à falhas (*Design and Simulation of Virtual Reconfigurable Circuit for a Fault Tolerant System*); esta é uma abordagem usando circuito virtual reconfiguravel que é

uma técnica muito semelhante a AVR para elaborar soluções de circuitos lógicos, com detecção de falhas nos elementos que compõem o circuito virtual reconfigurável.

2- Concepção e implementação de uma arquitetura reconfigurável virtual para diferentes aplicações de *hardware* evolutivo intrínseco (*Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware*); esta é uma abordagem que utiliza a técnica de AVR para elaborar soluções de circuitos reconhecedores de caracteres e filtros de imagem.

3.2 Projeto e simulação de circuito virtual reconfigurável para um sistema tolerante à falhas

Este artigo foi utilizado como base importante de contribuição para o presente trabalho, contribuindo com o conceito de circuito virtual reconfiguravel, onde circuitos eletrônicos são codificados por meio de sequencias de *bits* e são evoluídos para obter o resultado desejado. Cada circuito é configurado e simulado por um conjunto de vetores de entrada; suas saídas são comparadas aos valores esperados e um valor de aptidão é calculado para determinar o quanto este circuito atende à especificação e o processo é repetido gerando novos circuitos por meio da aplicação de operadores genéticos, até que um circuito atenda a especificação ou a quantidade de repetições limite seja atingida.

3.2.1 Circuito Virtual Reconfigurável

O uso de circuitos virtuais reconfiguráveis permite introduzir uma nova abordagem para o projeto de sistemas evolutivos completos em um único FPGA. A Figura 3.1, mostra que o circuito virtual reconfigurável consiste de uma matriz de elementos programáveis. Quando o circuito virtual reconfigurável é carregado no FPGA, seu *bitstream* de configuração cria uma matriz de elementos programáveis (EP) neste dispositivo. O VRC toma a forma de uma matriz bidimensional de elementos regulares programáveis, que é configurada com cada sequência obtida pela evolução e submetida aos testes para determinar sua aptidão.

EP é um elemento de *hardware* que pode ser configurado por uma entrada (Conf.), esta determina quais operações serão realizadas com os dados recebidos (In1, In2, In3, etc.).

Cada EP na primeira coluna tem suas entradas a partir de qualquer uma das entradas externas. Cada coluna usa como entrada as saídas da coluna anterior; e a última coluna fornece as saídas. Cada EP é configurada por um subconjunto de *bits* obtidos da configuração fornecida pela evolução.

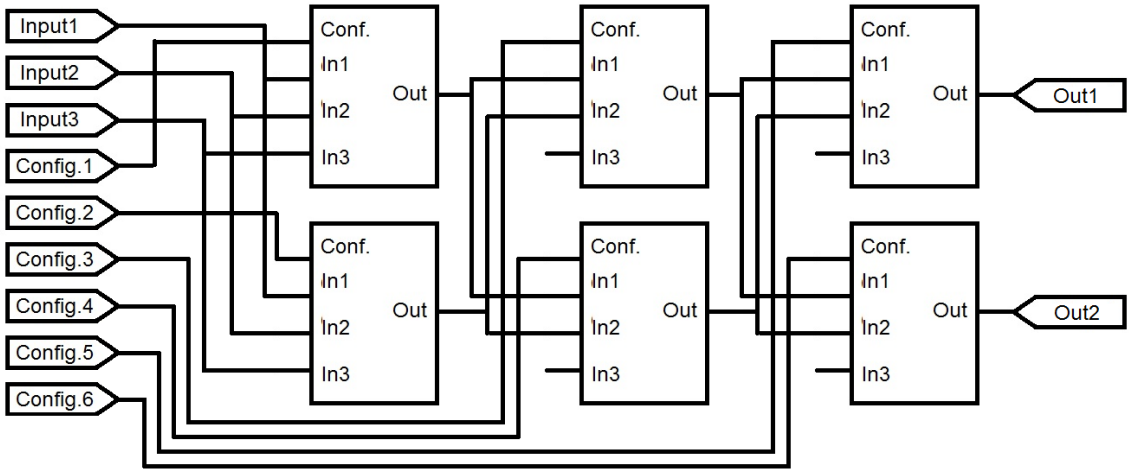

Figura 3.1 Disposição dos EPs dentro do VRC adaptado (CANCARE, BHANDARI, *et al.*, 2011).

3.2.2 Resultados

Neste artigo VRC é implementado na solução de dois problemas: um circuito somador e um circuito aleatório, utilizando uma matriz de 3 × 2 elementos, em ambos os casos o circuito atende as especificações, com uma ocupação da FPGA de 246/768 *slices*.

3.3 Concepção e implementação de uma arquitetura reconfigurável virtual para diferentes aplicações de *hardware* evolutivo intrínseco

Este artigo também é base de importante contribuição para o presente trabalho de mestrado, contribuindo com o conceito de AVR, onde elementos construídos por uma codificação na FPGA são configurados por meio de sequencias de *bits* e evoluídos para se obter o resultado desejado. Cada elemento é configurado e simulado por um conjunto de vetores de entrada; suas saídas são comparadas aos valores esperados e um valor de aptidão é calculado para determinar o quanto este circuito atende à especificação e o processo é repetido gerando novos circuitos por meio da aplicação de operadores genéticos, até que um circuito atenda à especificação ou a quantidade de repetições limite seja atingida.

3.3.1 Implementação do AVR

Nesse artigo, a implementação é feita em uma placa “Celoxica RC1000” com uma FPGA “Xilinx Virtex XCV2000E”. Dois são os métodos de transferência de dados do PC para a FPGA: dois caminhos de 8 *bits* unidirecionais chamados de controle e *status* da porta, e quatro bancos de memória de 32 *bits*; esta organização pode ser vista na Figura 3.2, composta por:

- Unidade da Matriz composta pelos elementos de função, onde a configuração é implementada;
- Unidade EA, responsável pelo algoritmo evolutivo, onde ocorre a evolução;
- Unidade de aptidão, que realiza a avaliação das configurações enviadas para a unidade da matriz;
- Unidade de controle e interface responsável por enviar e receber os dados do PC;

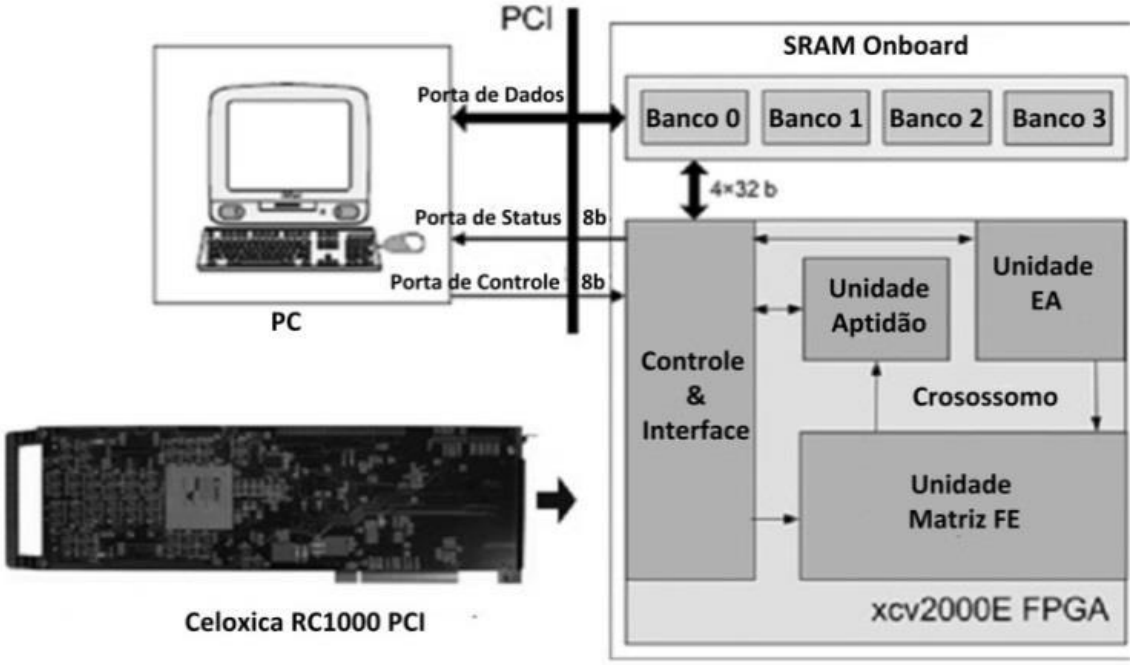

Figura 3.2 Organização do AVR, adaptado (WANG, CHEN e LEE, 2007).

3.3.2 Resultados

O artigo apresenta duas aplicações: reconhecimento de caracteres e filtro de imagens. Apenas a aplicação de filtro de imagens é descrita neste trabalho de mestrado, por apresentar resultados compatíveis para fins de comparação. Este filtro é aplicado no processamento de imagens que tiveram dois tipos de ruído adicionados: o ruído Gaussiano (com média 0 e variância 0,008) e o ruído Sal e Pimenta (com 5% de *pixels* corrompidos); os ruídos foram gerados usando funções do MatLab. Duas soluções de filtro foram geradas usando, para o treinamento a imagem Lena, em escala de cinza de 8 *bits* por *pixel* e dimensão de 256 × 256 *pixels*.

Assim, foram executadas 100 evoluções, como configuração experimental, e foi usado a MDPP (média da diferença absoluta por *pixel*) para comparar a imagem filtrada com a imagem original sem ruído.

Um exemplo do operador da imagem evoluída para o processamento do ruído Sal e Pimenta é mostrado na Figura 3.3. Tal operador é composto por nove EFs e emprega três funções: F3, F4 e F6.

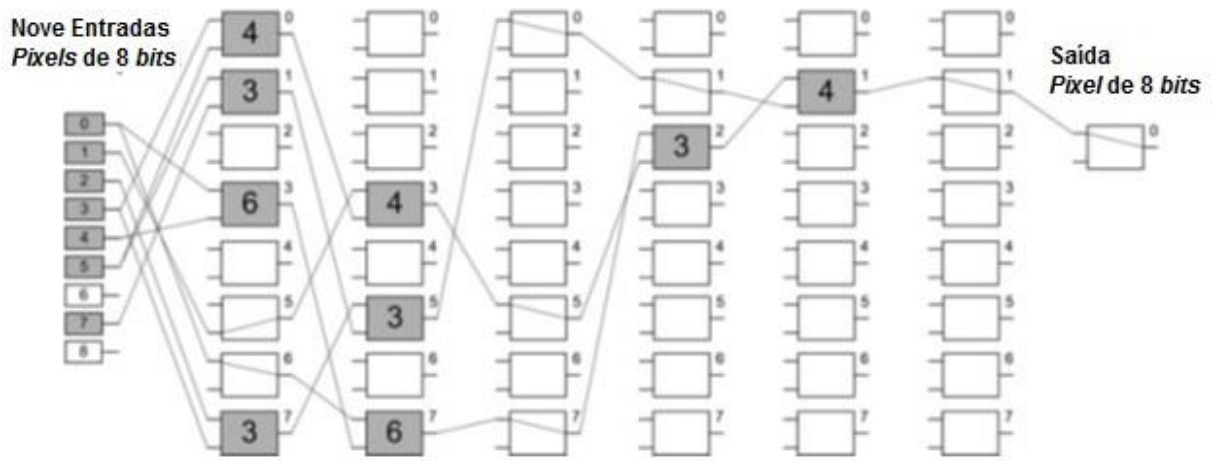

Figura 3.3 Exemplo do operador, adaptado (WANG, CHEN e LEE, 2007).

3.4 Considerações finais

Na maioria dos trabalhos encontrados, a aplicação ficou restrita aos circuitos digitais, tais como: somadores, multiplicadores, geradores de paridade, etc. E muitos desses, utilizaram como processo de evolução o AG.

Neste trabalho de mestrado, a proposta é implementada tanto em circuitos digitais como em filtro de imagens, demonstrando-se sua flexibilidade na solução de problemas.

Capítulo 4

PROPOSTA DO TRABALHO

Neste capítulo, aborda-se a metodologia e implementação de uma arquitetura flexível e reconfigurável com simulações em geração de circuitos lógicos e processamento digital de imagens, demonstrando a preparação da ferramenta GPLAB, o hardware reconfigurável utilizando AVR híbrido e apresentando os resultados.

4.1 Metodologia e Implementação

Neste trabalho, pretende-se utilizar a ferramenta GPLAB para obter soluções lógicas aplicáveis a sistemas embarcados; as soluções são convertidas em uma saída semelhante ao genótipo/fenótipo da programação genética cartesiana (PGC), podendo ser aplicada diretamente a sistemas embarcados utilizando a AVR híbrida.

Essa AVR é considerada híbrida por usar um HE construído com uma matriz de EFs em uma FPGA, como é utilizado usualmente por HE intrínseca, e que vão receber sua configuração de uma solução gerada pela ferramenta GPLAB, como em um HE extrínseco.

PG é uma forma eficiente de encontrar soluções para serem aplicadas em sistemas embarcados, e a GPLAB é uma ferramenta eficiente de PG.

Para demonstrar a proposta, este método é utilizado em uma simulação de geração autônoma de circuitos lógicos para encontrar um programa adequado para solucionar uma determinada relação entre entradas e saída, e também, em uma aplicação de processamento de imagem, que encontre uma forma de aplicar filtros para se obter um determinado resultado. Essas soluções são representadas em um formato possível de ser empregado diretamente em um sistema embarcado usando a

tecnologia de FPGA. Os resultados deste trabalho de geração automática de Circuitos Lógicos e Filtros de Imagem são comparados com o estudo de (VASU, SRIVASTAVA, *et al.*, 2014) para solução de circuitos; e para solução de filtros são comparados com o estudo de (WANG, CHEN e LEE, 2007), que apresentam bons resultados para aplicação de técnicas de HE. Essas simulações têm o objetivo de demonstrar a eficiência e a flexibilidade da proposta.

4.1.1 *Hardware* Evolutivo Híbrido Proposto

Neste trabalho, uma plataforma em HE é proposta utilizando AVR, descrita em Verilog HDL como uma segunda camada de reconfiguração, como em HE intrínseco. Nesta plataforma a evolução do circuito é feita em um computador, como em HE extrínseco, utilizando uma ferramenta de programação genética, e apenas a melhor solução é usada para reconfigurar o *hardware*. Esta solução é hibrida por utilizar características de HE intrínseca e extrínseca.

Com isso há alguns benefícios: menor comprimento do cromossomo, tempo de reconfiguração reduzido, facilidade para executar testes no *hardware* durante as evoluções, códigos descritos em Verilog HDL permitindo ser utilizado em diversas plataformas e circuito flexível, podendo utilizar uma matriz otimizada.

4.1.1.1 Mapeamento do genótipo e fenótipo

Neste trabalho, o fenótipo é uma matriz bidimensional de EFs. Esta matriz é constituída por linhas e colunas, sendo que os EFs de uma coluna podem ser conectados a qualquer EF da coluna anterior. EFs da primeira coluna podem ser conectadas às entradas e a última coluna pode ser conectada às saídas. Cada EF tem suas funções pré-definidas em um conjunto de funções básicas capazes de gerar uma solução. Estas funções podem ser de qualquer tipo, por exemplo: AND, NAND, OR, NOR, XOR, adição, subtração, funções especiais, etc.

O genótipo é um conjunto de cadeias de *bits* lineares e binários, onde cada cadeia define as funções e as conexões de cada EF na matriz, como apresentado na Figura 4.1. Essa cadeia é um gene do cromossomo e está dividida em duas partes, a primeira contém a origem dos operandos que são utilizados pelo EF e a segunda contém a função que o EF executa. Semelhantemente ao que ocorre na PGC, onde um cromossomo também configura uma matriz bidimensional. Essa configuração é

detalhada nos próximos tópicos. O conjunto de cadeias de *bits* é armazenado em uma memória do HE/FPGA.

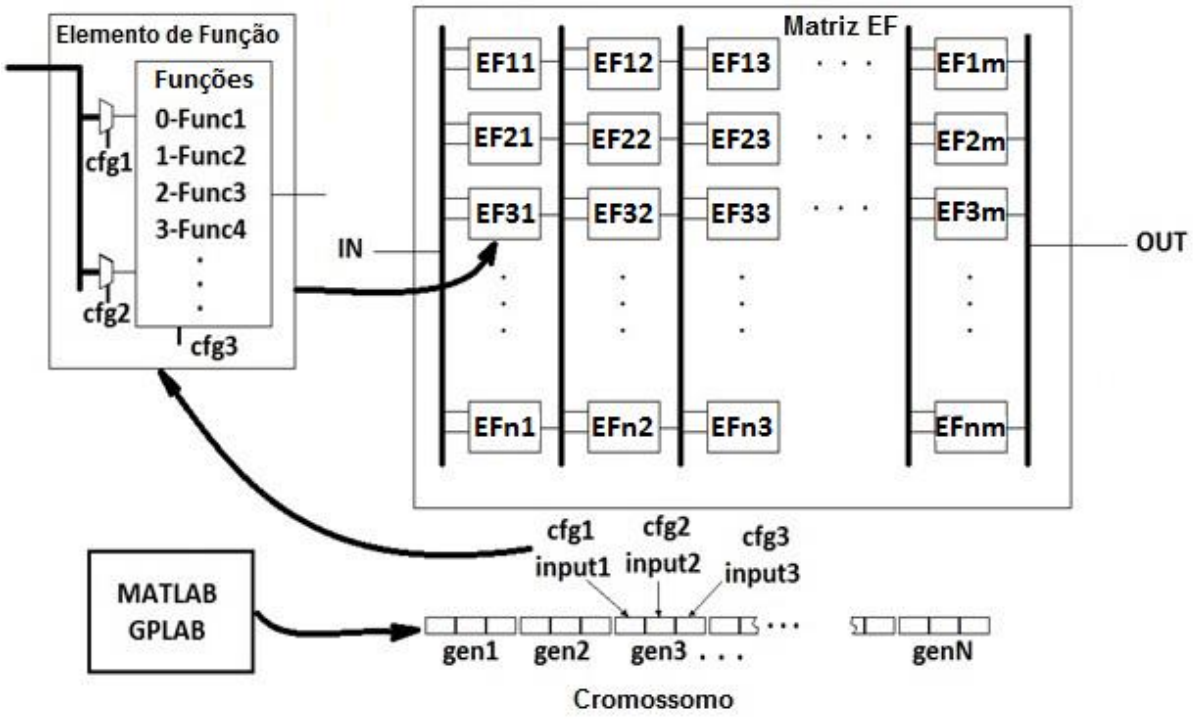

Figura 4.1 Mapeamento genótipo-fenótipo.

4.1.1.2 Programação Genética

Neste trabalho, a programação genética é executada em um computador utilizando a ferramenta GPLAB para MATLAB.

O uso da GPLAB traz muitos benefícios: é uma ferramenta testada, é de fácil configuração, apresenta grande versatilidade e atende os principais recursos necessários na elaboração de uma evolução, podendo ser facilmente adaptada a diversos casos. No entanto, a ferramenta apresenta alguns inconvenientes: o melhor cromossomo é apresentado em formato de árvore, *string* ou estrutura, e por ser uma programação genética convencional, não é possível formar indivíduos com mais de uma saída.

4.1.1.3 Elaboração do *Hardware*

O *hardware* é composto por uma placa de desenvolvimento contendo uma FPGA da Altera (ALTERA CORPORATION, 2007). Na elaboração dos circuitos é usado um computador rodando o *software* Quartus II da Altera, onde são

desenvolvidos os códigos em Verilog HDL e carregados na FPGA. A ferramenta de evolução é o GPLAB rodando nesse mesmo computador e *scripts* em MATLAB são usados para converter os resultados para o formato de memória da FPGA, conforme a Figura 4.2.

Para a FPGA é criado um código Verilog HDL, como um módulo de controle para realizar o controle da matriz de EFs. Este vai manipular a memória e comunicar com o computador para obter o resultado da evolução. Este resultado é armazenado na memória, a qual é lida pelo modulo de controle para configurar cada uma das EFs. A elaboração do *hardware* é detalhada nos próximos tópicos.

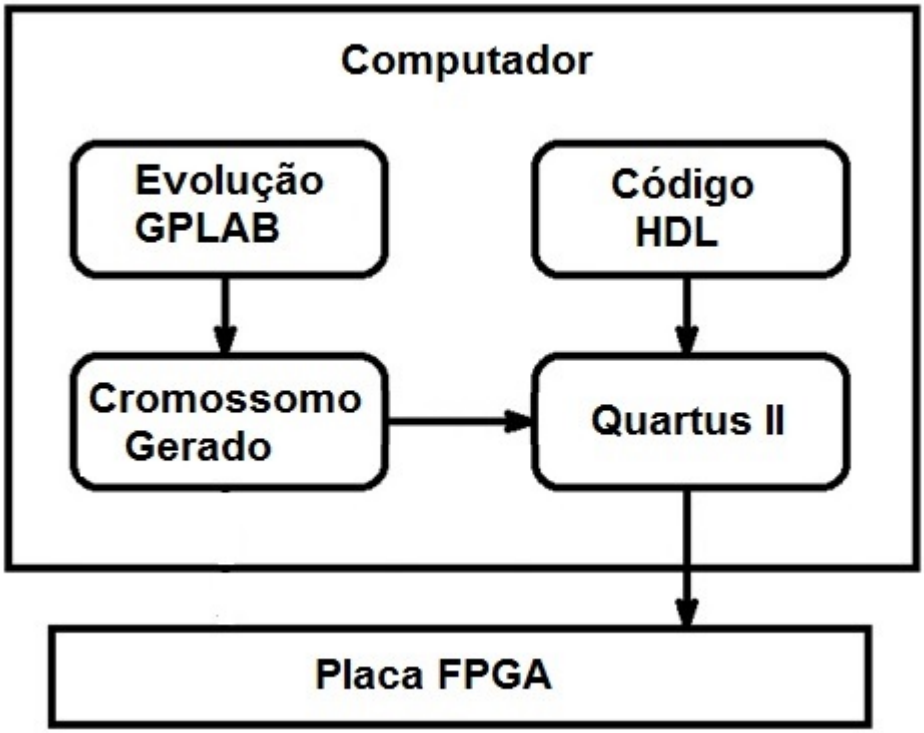

Figura 4.2 Organização do *hardware.*

4.1.1.4 Construção das Funções Elementares

Para a geração do *hardware* é preciso elaborar um circuito básico capaz de atender às funções e ser conectado aos terminais definidos pela programação genética.

Este circuito básico é chamado de elemento de função (EF); esses EFs são gerados na FPGA como uma matriz cartesiana que é interligada conforme o mapeamento de memória gerado.

A EF precisa atender as funções e estabelecer as ligações entre funções e terminais conforme foram gerados na programação genética. No caso de se criar em circuitos lógicos, por exemplo, a EF deve exercer as funções lógicas NAND, NOR, AND e OR, ou outras que sejam adequadas, como apresentado na Figura 4.3. A configuração de cada EF é realizada através de um código que chamaremos de OPCODE e cada EF recebe seu OPCODE que está armazenado no mapeamento da memória. O *hardware* do FE, nesse exemplo, é criado contendo quatro funções lógicas. Essas funções recebem seus operandos através da seleção de dois MUXs e as funções são selecionadas por um MUX de saída. O MUX é um multiplexador e funciona como uma chave que seleciona sua posição conforme uma configuração.

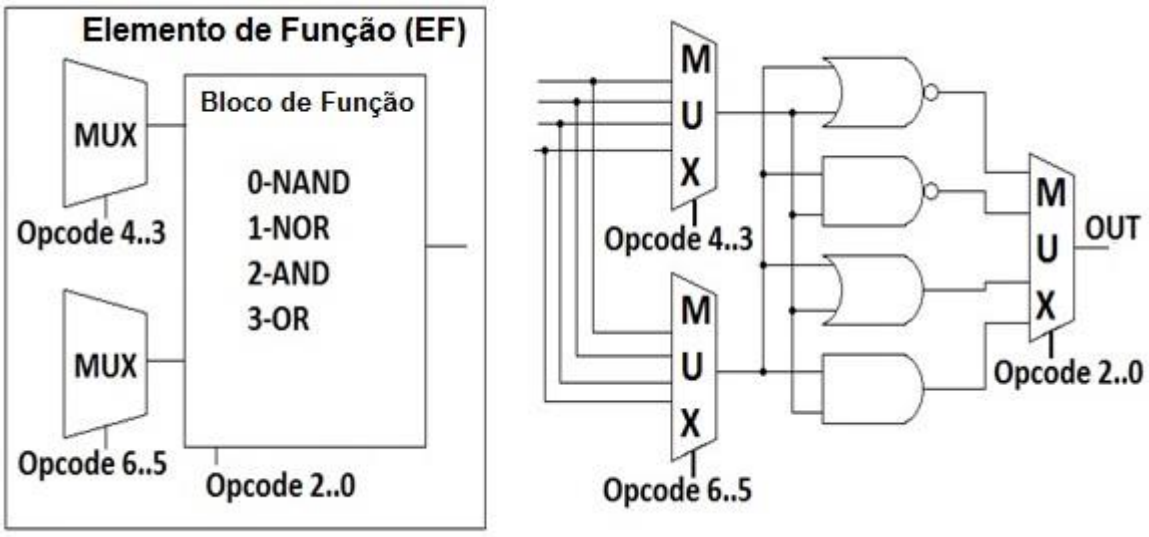

Figura 4.3 Elemento de função (EF) proposto para circuitos lógicos.

4.1.1.5 Construção do controle da matriz

O *hardware* é um conjunto de EFs dispostos em uma matriz cartesiana, de tamanho configurável, que possibilita a reconfiguração de suas ligações através da leitura de dados de uma memória que contém o OPCODE de cada EF, realizando a reconfiguração conforme o cromossomo selecionado. Um código Verilog HDL é responsável pela criação dessa matriz, com as dimensões adequadas ao resultado de cada evolução, bem como de um controle que faz a leitura dos dados do cromossomo armazenado na memória, como apresentado na Figura 4.4.

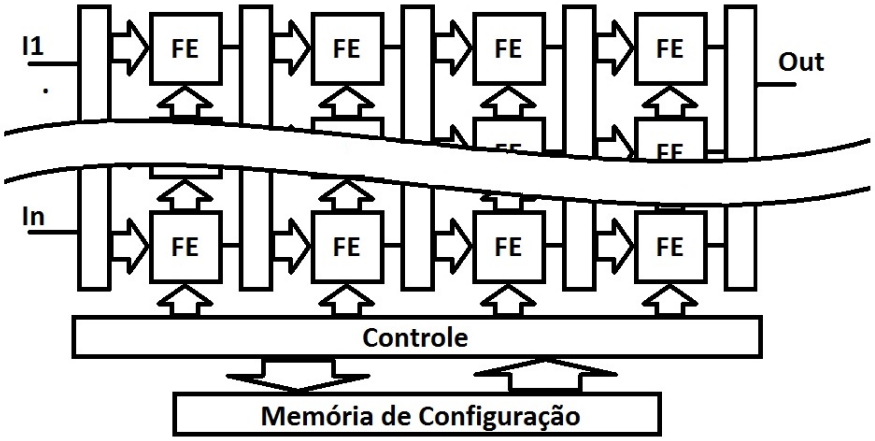

Figura 4.4 Esquemático da matriz de EFs.

4.1.1.1 Gerando Mapeamento

A ferramenta GPLAB gera um resultado na forma de uma árvore que contém o melhor indivíduo.

Uma função de conversão é executada e converte a árvore em uma representação de suas ligações, lendo a *string* que representa tal árvore.

Esta função lê a *string* que a GPLAB apresenta como melhor indivíduo; em primeiro lugar, encontra os nós com terminais, e na sequência encontra os nós internos, lendo a árvore da esquerda para direita e de baixo para cima, montando uma sequência de mapeamento de memória que é usada para configurar o *hardware*. Isso é semelhante a dizer que, girando o gráfico da árvore em 90º no sentido horário, tem-se o gráfico de uma PGC, conforme apresentado na Figura 4.5.

Dessa forma, gera-se uma descrição do indivíduo conforme é disposto na matriz de EFs, contendo a posição da EF, função para executar e operadores. Essa descrição é convertida para um cromossomo. Este cromossomo é um conjunto de várias sequências de *bits*. Cada sequência é um gene, e cada gene é arquivado na memória da AVR, e a posição da EF torna-se a posição de memória de cada gene. O formato do gene é uma sequência de *bits* que está dividido em duas partes; a primeira contém a identificação dos operandos e a segunda a função para executar. Seu tamanho é flexível e é adaptada para o tipo de *hardware* que se deseja implementar na EF.

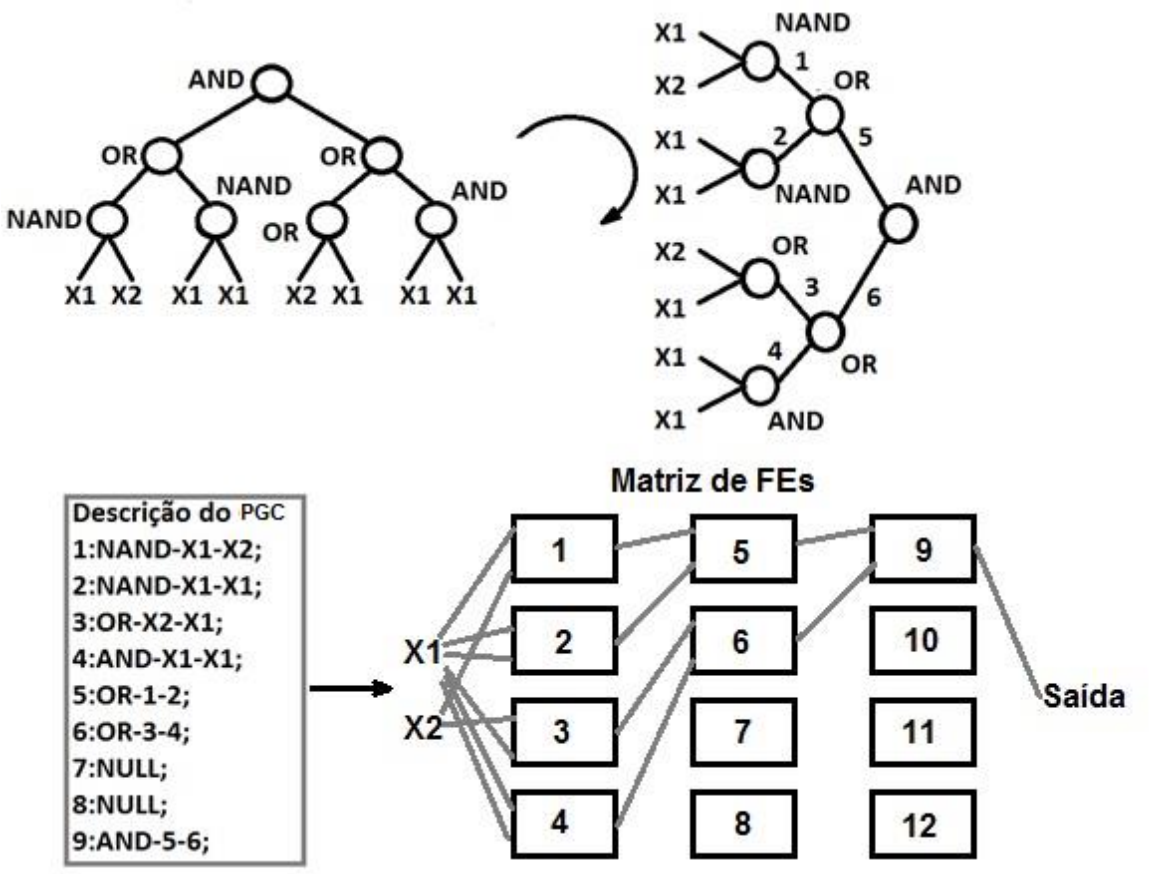

Figura 4.5 Exemplo de árvore e descrição do PGC.

4.1.2 Sistema para Geração de Circuitos Lógicos

Desde o princípio do uso de circuitos lógicos digitais, que se tornou possível com o surgimento de circuitos eletrônicos, muitos estudos foram feitos para desenvolver uma forma prática de elaborar um circuito que atenda a uma relação de entrada/saída desejada. A utilização de programação genética para encontrar tais circuitos se mostra eficiente para executar tal elaboração.

Como exemplo, a GPLAB é preparada para fornecer as soluções para os circuitos lógicos a seguir, em um formato de dados capaz de ser enviado à AVR da FPGA. A GPLAB é configurada para solucionar os circuitos lógicos: ‘XOR’, ‘ADD’ e ‘Random’. A relação entrada/saída é apresentada na Tabela 4.1.

A AVR é preparada para solucionar qualquer um dos circuitos lógicos apresentados. Um EF é criado conforme descrito na Figura 4.3, contendo funções lógicas NAND, NOR, AND e OR para gerar um circuito capaz de resolver tais lógicas.

Tabela 4.1 – Lógica desejada.

Add				Random				Xor		
In			Out	In		Out		In		Out
A	B	C	O	A	B	C	O	A	B	O
1	0	0	1	1	1	0	1	0	0	0
0	1	0	1	0	0	0	0	0	1	1
0	0	0	0	1	1	1	0	1	0	1
1	1	1	1	0	1	1	0	1	1	0

4.1.2.1 Configuração da ferramenta GPLAB

Nesta simulação são criadas as funções e demais processos necessários ao uso da GPLAB na geração de circuitos lógicos, elaborando uma nova demonstração para essa ferramenta, além de realizar um levantamento de resultados e comparar esses com o artigo de (VASU, SRIVASTAVA, *et al.*, 2014), comprovando que o uso da ferramenta GPLAB e o VRA Híbrido proposto aqui são adequados e podem ser utilizados nos estudos de geração de circuitos lógicos.

Para isso, a ferramenta GPLAB é adequada a um processo de geração de circuitos lógicos para alguns casos; criar uma função principal de configuração e execução da ferramenta; criar as funções genéticas, operadores genéticos e terminais adequados para um circuito lógico capaz de atender as relações apresentadas.

São criados os ambientes para o levantamento de dados, a execução e o processo genético para registrar os dados. Também, são avaliados os dados determinando o quanto esses são capazes de produzir soluções adequadas. Por fim, os resultados obtidos são comparados com outros resultados da literatura.

4.1.2.2 Preparação da Ferramenta GPLAB

Os operadores genéticos criados para resolução do problema são:

- '**NAND':** função que executa a operação NAND com seus operadores.
- '**NOR':** função que executa a operação NOR com seus operadores.

Outros operadores genéticos já disponíveis no MATLAB foram usados para resolução do problema:

- '**AND':** função que executa a operação AND com seus operadores.
- '**OR':** função que executa a operação OR com seus operadores.

A PG é executada com os seguintes operadores de recombinação: 'crossover', de 2 pais, de dois filhos e 'mutation', de 1 pai, de 1 filho; funções: NAND, NOR, AND e OR, e criação dos indivíduos pelo método de crescimento ‘fullinit’**;** a seleção dos indivíduos é do tipo ‘roulette’; as árvores são simétricas de profundidade 3, com população de 150 indivíduos e 20 gerações, com população fixa e mantendo-se o melhor indivíduo de cada geração.

Para o cálculo da aptidão, o método utilizado é o padrão do GPLAB, onde este calcula para cada indivíduo a soma da diferença absoluta entre todos os valores dos casos de aptidão esperados e todos os valores retornados pelos indivíduos em todos os casos de aptidão. Os melhores indivíduos são os que retornam valores menores.

4.1.2.3 Descrição dos Testes

Em cada um dos casos descritos na Tabela 4.1, a PG é executada pela GPLAB por 30 vezes e são coletados: *ID* do melhor indivíduo, *Geração* em que ocorreu a solução e o tipo de operador que gerou o melhor indivíduo.

É montada uma planilha de valores e um gráfico que demonstra a probabilidade de encontrar uma solução em cada geração.

4.1.2.1 Execução

Após cada evolução realizada pela ferramenta GPLAB é obtido um resultado para cada geração de circuito lógico, conforme demonstra a Figura 4.6, e a conversão é realizada obtendo-se cada cromossomo para ser aplicado ao *hardware*, apresentado na Figura 4.7.

A conversão é realizada obtendo-se cada cromossomo para ser aplicado ao *hardware*, conforme apresentado na Figura 4.7. A memória é carregada com o cromossomo da melhor solução, cada posição da memória corresponde a um determinado EF e os dados dessa posição de memória configura o respectivo EF. Um EF é codificado usando-se 7 *bits*, 2 + 2 *bits* (CFG 1 e CFG 2), que são utilizados para controlar as seleções de entradas de cada EF; 3 *bits* (CFG 3) são usados para selecionar a função implementada.

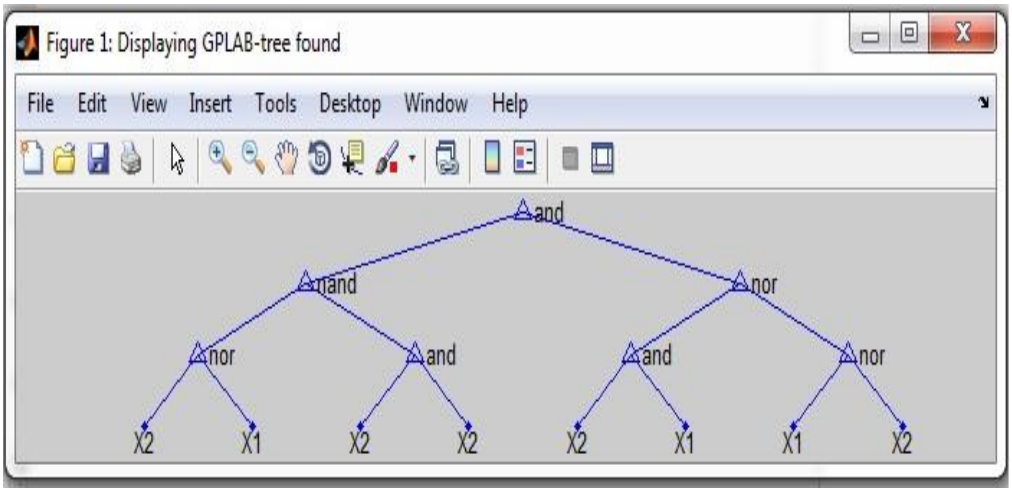

Figura 4.6 Exemplo de resultado obtido pela GPLAB.

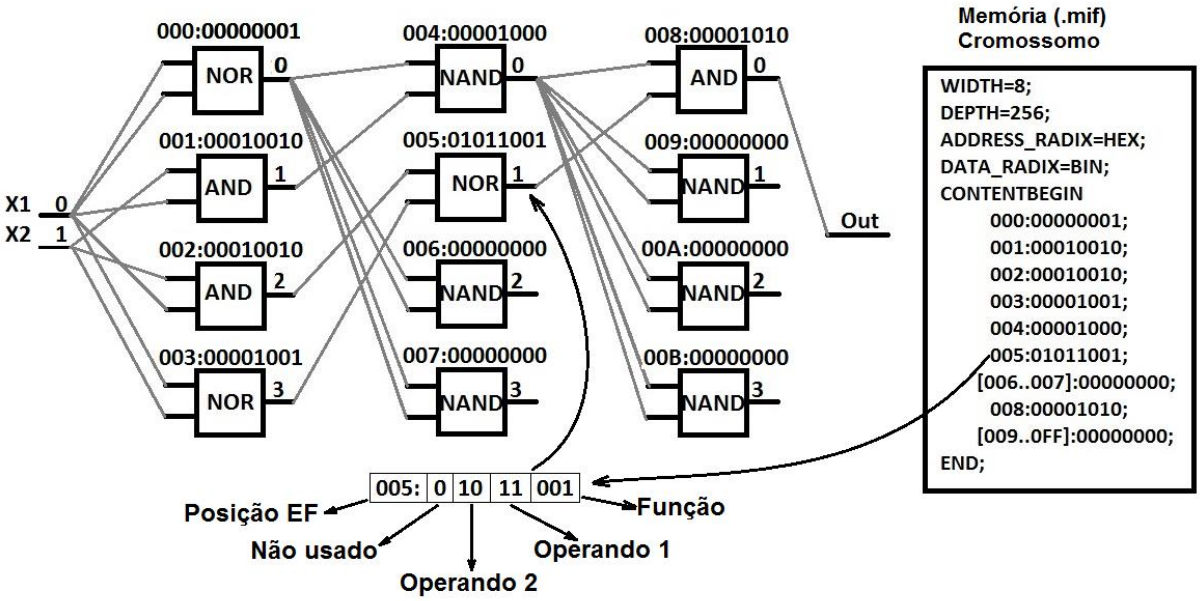

Figura 4.7 Modelo do fenótipo que se deseja implementar.

4.1.3 Simulação da Aplicação em Processamento Digital de Imagens

Nesta simulação, são criadas as funções e demais processos necessários ao uso da GPLAB no processamento digital de imagens, utilizando-o mesmo filtros de ruídos adicionados às imagens, elaborando uma nova demonstração para essa ferramenta. Também, realiza-se um levantamento de resultados, comprovando que o

uso da ferramenta GPLAB é adequado e que pode ser utilizada nos estudos de processamento digital de imagens e desenvolvimento de *hardware*.

A AVR é preparada para solucionar e encontrar filtros de imagens. A ferramenta GPLAB é configuada para encontrar soluções de filtros para ruídos do tipo Sal e Pimenta e Gaussiano, utilizando-se técnicas do domínio espacial e de morfologia matemática para encontrar o filtro apropriado, demonstrando-se assim sua flexibilidade, como apresentado na Figura4.8.

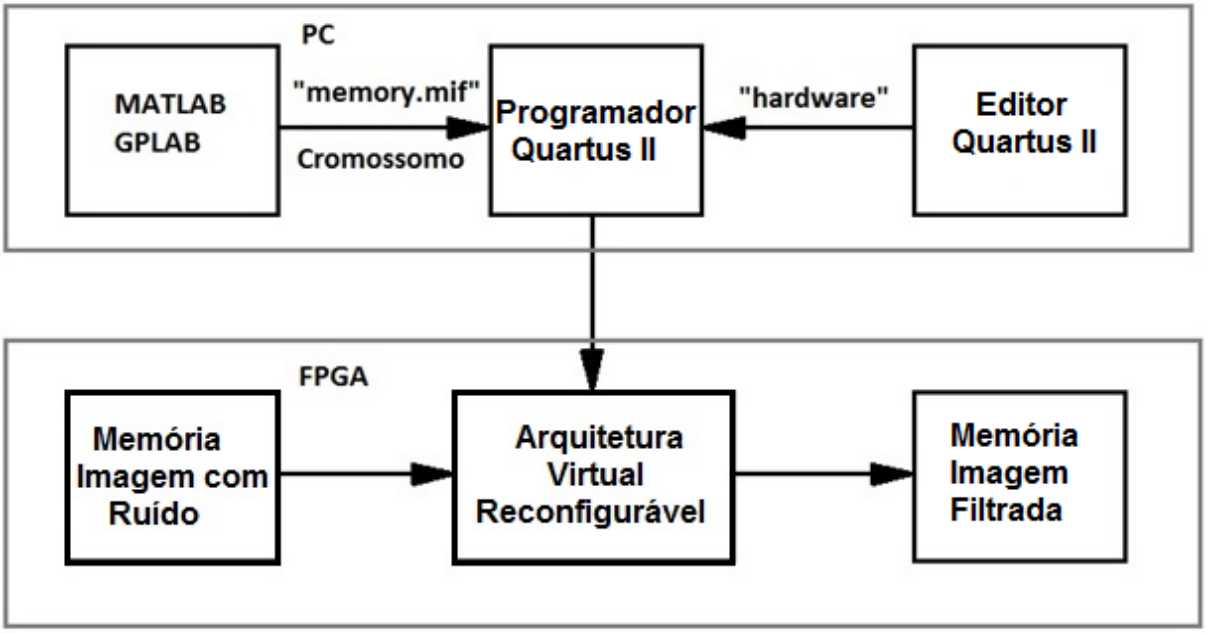

Figura 4.8 Sistema proposto.

O filtro de imagem no dominio espacial é empregado com frequência nas implementações baseadas em HE, uma vez que seu custo de *hardware* é menor do que o filtro de imagem no dominio da frequência.

Diferentes abordagens apresentam sucesso na evolução de filtros de imagens, em diferentes ruídos adicionados, tais como ruídos Sal e Pimenta e Gaussiano, utilizando-se diversos operadores no dominio espacial. Neste trabalho, são utilizados 8 funções executando tais operadores, como demonstrado na Tabela 4.2.

Esses operadores são equivalentes às funções: ImageMinAB, ImageMaxAB, ImageXorAB, ImageALeftShift, ImageAddABShift, ImageAddAB1Shift, ImageB, ImageA . Essas funções são usadas na evolução dos filtros pelo GPLAB e são descritas nos próximos tópicos.

Tabela 4.2 – Operadores espaciais de imagem.

Index	Function	Index	Function
0	*a*	4	*Min(a,b)*
1	*(a+b)>>1*	5	*a<<1*
2	*(a+b+1)>>1*	6	*Xor(a,b)*
3	*Max(a,b)*	7	*b*

A GPLAB é preparada para fornecer as soluções para os filtros de imagens. São utilizados para os testes as imagens de “lena” e “cameraman”, ambas de 256×256 *pixels* de 8 *bits* em escala de cinza, adicionados os ruídos Sal e Pimenta (*Salt & Pepper*) e Gaussiano (*Gaussian*), usando as respectivas funções do MATLAB. A ferramenta fornece um formato de dado capaz de ser enviado à AVR da FPGA.

Um EF é criado contendo as 8 funções que são capazes de gerar um filtro para a imagem, como apresentado na Figura 4.9.

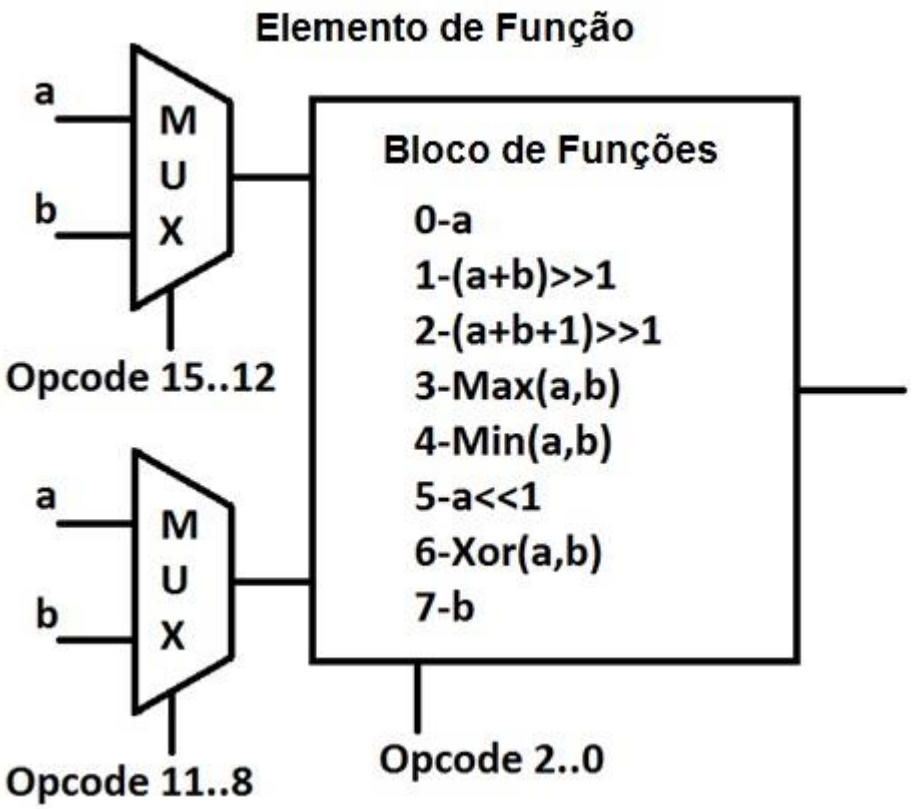

Figura 4.9 Elemento de função proposto para filtragem de imagens.

Neste trabalho, cada EF resulta em um operador de filtro de imagem, no qual este corresponde a um circuito digital de nove entradas com 8 *bits* e uma única saída

de 8 *bits*, que processa a imagem em escala de cinza de 8 *bits* por *pixel*. Cada *pixel* da imagem filtrada é gerado por uma janela 3×3 com os *pixels* mais próximos daquele convertido da imagem com ruído. O processo de filtragem ocorre para cada *pixel* da imagem original (ruidosa), utilizando-se os seus oito vizinhos na imagem de entrada para efetuar o cálculo, como apresentado na Figura 4.10.

Em contraste com a evolução do sistema de circuitos lógicos, em que as funções são baseadas em portas lógicas (NAND, NOR, AND E OR) e a saída é de apenas1 *bit*, a abordagem da evolução de filtro de imagem é utilizada em nível funcional com as operações de máximo, mínimo, média, etc, e dados de 8 *bits* são empregados na matriz de EF.

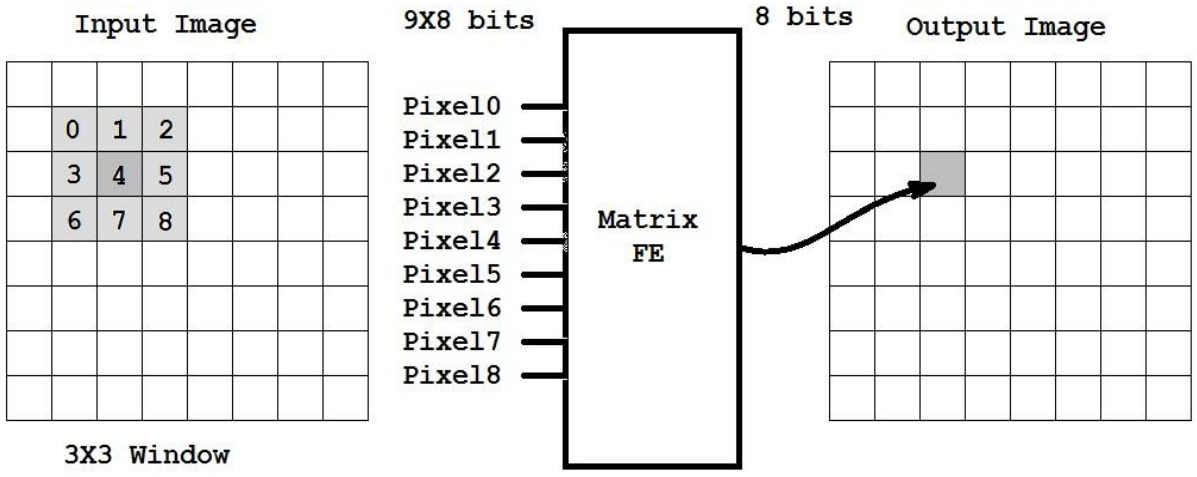

Figura 4.10 Processamento da imagem por janela 3X3.

No entanto, para o caso de filtro de imagens, EFs da primeira coluna recebem suas entradas que são um dos 9 *pixels* da janela, onde esses *pixels* são obtidos de uma função em Verilog HDL, ou seja, de um *hardware* que faz a separação dos *pixels*. Esta função faz uma leitura sequencial da memória da FPGA, que contém a imagem com ruído e armazena os dados em registros; a quantidade de registros necessários são em número de duas linhas mais três *pixels* dessa imagem. Portanto, esta função é constituída por um conjunto de registros formando um registrador do tipo *First In First Out* (FIFO), sendo *T* o tamanho total deste e *L* o número de *pixels* por linha; o tamanho é obtido conforme a Equação a seguir:

$$T = 2 * L + 3 \tag{9}$$

A função faz a leitura sequencial dos *pixels* da imagem e segue carregando a entrada do registrador FIFO, onde esses vão sendo deslocados até todo o registrador conter os dados da imagem. Neste momento, tem-se a primeira janela 3X3 disponível e assim sucessivamente. Quando cada janela 3X3 é completada, esta função vai disponibilizando esses *pixels* na entrada da matriz de EF, em conjunto com o endereço correspondente ao *pixel* de saída, como apresentado na Figura 4.11.

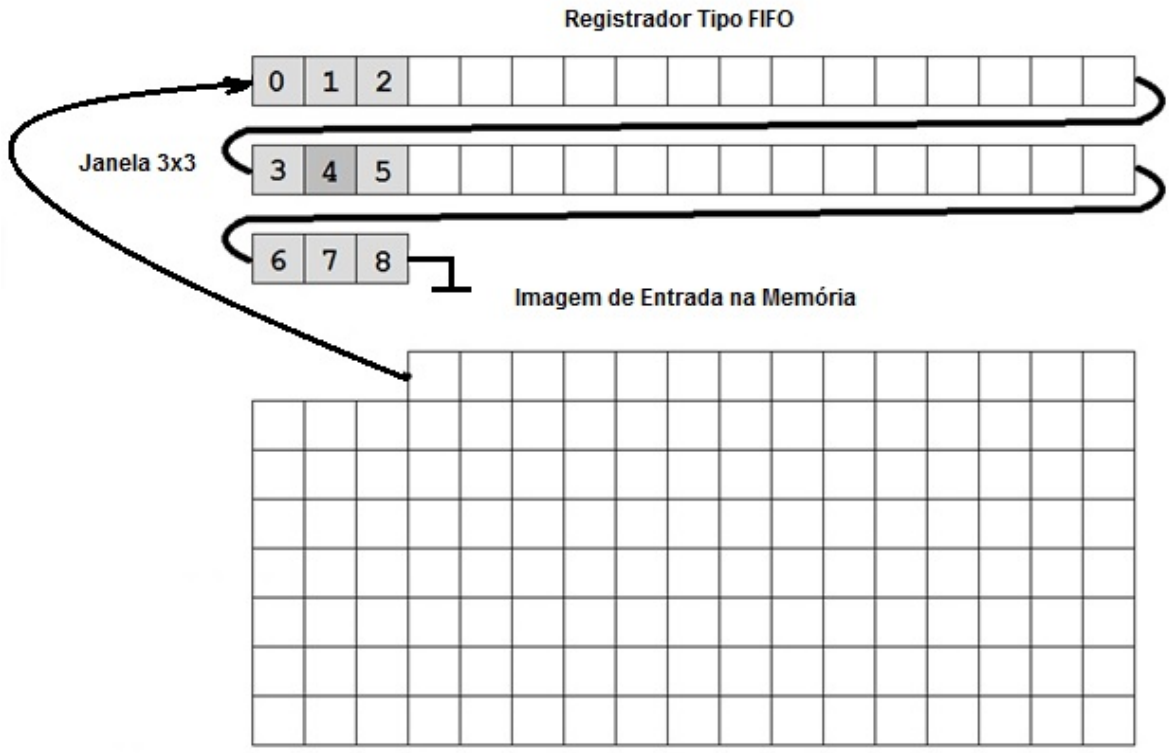

Figura 4.11 Processo para obter janela 3×3.

Um único EF é codificado usando-se 11 *bits*, 4 + 4*bits* (CFG 1 e CFG 2), que são utilizados para controlar as seleções de entradas de cada EF; 3 *bits* (CFG 3) são usados para selecionar a função implementada. O gene é composto por 16 *bits* para manter um padrão convencional em memórias, que geralmente usam dados em tamanhos de 8, 16, 32, 64, etc, *bits*. Com isso o código Verilog HDL não necessita ser muito modificado *para* atender outros processos de filtragem.

4.1.3.1 Configuração da ferramenta GPLAB

A ferramenta GPLAB é configurada como um caso de GP com operadores: 'crossover',de 2 pais, de dois filhos; 'mutation', de 1 pai, de 1 filho; funções:

ImageMinAB, ImageMaxAB, ImageXorAB, ImageALeftShift, ImageAddABShift, ImageAddAB1Shift, ImageB, ImageA, árvores simétricas de profundidade 3, população de 200 indivíduos e 30 gerações em cada evolução. Esse processo é detalhado nos próximos tópicos.

4.1.3.2 Cálculo da aptidão

O método utilizado para o cálculo da aptidão tem o objetivo de determinar a diferença entre a imagem filtrada e a imagem original e não corrompida. Para medir a qualidade da imagem filtrada, a diferença média por *pixel* (*mean difference per pixel* - MDPP) é implementada para o cálculo de custo (WANG, CHEN e LEE, 2007), sendo o tamanho original da imagem de *k X k*, onde somente a parte da imagem de *(k-2) X (k-2)* é considerada, porque os *pixels* nas fronteiras de imagem não são tratados para uma janela 3 X 3. O cálculo do custo é realizado de acordo com a equação abaixo:

$$\text{Custo}_{MDPP} = \sum_{i=1}^{k-2}\sum_{j=1}^{k-2}|v(i,j) - w(i,j)| \tag{10}$$

O custo é calculado somando-se a comparação da diversidade de cada *pixel* na imagem filtrada *v* com os *pixels* correspondentes *w* na imagem original e não corrompida.

4.1.3.3 Preparação da Ferramenta GPLAB

Os operadores genéticos criados para resolução do problema são:

- **'ImageMinAB':** função que seleciona o menor entre os seus dois operandos.
- **'ImageMaxAB':** função que seleciona o maior entre os seus dois operandos.
- **'ImageXorAB':** função que executa XOR *bit* a *bit* entre os seus dois operandos.
- **'ImageALeftShift':** função que executa um deslocamento à esquerda dos *bits* do operando A.
- **'ImageAddABShift':** função que executa a soma de seus dois operandos e desloca o resultado à direita, ou seja, divide por 2.

- **'ImageAddAB1Shift':** função que executa a soma de um ou mais de seus dois operandos e desloca o resultado à direita, ou seja, divide por 2.
- **'ImageB':** função que seleciona o operando B.
- **'ImageA':** função que seleciona o operando A.

Esses operadores permitem que se possa gerar as operações morfológicas para realizar a filtragem da imagem.

Os terminais genéticos criados para resolução do problema são:

- **'ImagePixel0':** função que carrega os 254×254 *pixels* equivalentes ao *pixel* 0 da janela na imagem com ruído.
- **'ImagePixel1':** função que carrega os 254×254 *pixels* equivalentes ao *pixel 1* da janela na imagem com ruído.
- **'ImagePixel2':** função que carrega os 254×254 *pixels* equivalentes ao *pixel* 2 da janela na imagem com ruído.
- **'ImagePixel3':** função que carrega os 254×254 *pixels* equivalentes ao *pixel* 3 da janela na imagem com ruído.
- **'ImagePixel4':** função que carrega os 254×254 pixels equivalentes ao *pixel* 4 da janela na imagem com ruído.
- **'ImagePixel5':** função que carrega os 254×254 pixels equivalentes ao pixel 5 da janela na imagem com ruído.
- **'ImagePixel6':** função que carrega os 254×254 pixels equivalentes ao pixel 6 da janela na imagem com ruído.
- **'ImagePixel7':** função que carrega os 254×254 pixels equivalentes ao pixel 7 da janela na imagem com ruído.
- **'ImagePixel8':** função que carrega os 254×254 pixels equivalentes ao pixel 8 da janela na imagem com ruído.

Essas funções terminais executam uma leitura da imagem com ruído de forma que o *pixel* equivalente da janela em processamento esteja corretamente disponível.

A PG é executada com operadores de recombinação '*crossover'* e mutação '*mutation'*, e pela criação dos indivíduos pelo método de crescimento '*growinit'*; a seleção dos indivíduos é do tipo '*roulette'*, com população fixa e mantendo-se o melhor indivíduo a cada geração.

A ferramenta GPLAB é configurada como um caso de GP com operadores de *crossover* de 2 pais e dois filhos; *mutation* de 1 pai e 1 filho; e as seguintes funções: ImageMinAB, ImageMaxAB, ImageXorAB, ImageALeftShift, ImageAddABShift, ImageAddAB1Shift, ImageB, ImageA, árvores simétricas de profundidade 3, criação dos indivíduos pelo método de crescimento '*fullinit*', onde a seleção dos indivíduos é do tipo '*roulette*', com população de 200 indivíduos e 30 gerações em cada evolução.

Para o cálculo da aptidão, o método utilizado é o da aptidão MDPP já descrito. Os melhores indivíduos são os que retornam valores de MDPP menores, ou seja, não muito diferentes dos valores esperados.

4.1.3.1 Descrição dos Testes

Em cada um dos ruídos adicionados descritos, a PG é executada pela GPLAB 100 vezes e são coletados: ID do melhor indivíduo, geração em que ocorreu a solução, índice MDPP e o tipo de operador que gerou o melhor indivíduo.

4.1.3.1 Execução

Após cada evolução realizada pela ferramenta GPLAB é obtido um resultado para o filtro de imagem, como apresentado na Figura 4.12, e a conversão é realizada obtendo-se cada cromossomo, em arquivo de memória, para ser aplicado ao *hardware*, como apresentado na Figura 4.13.

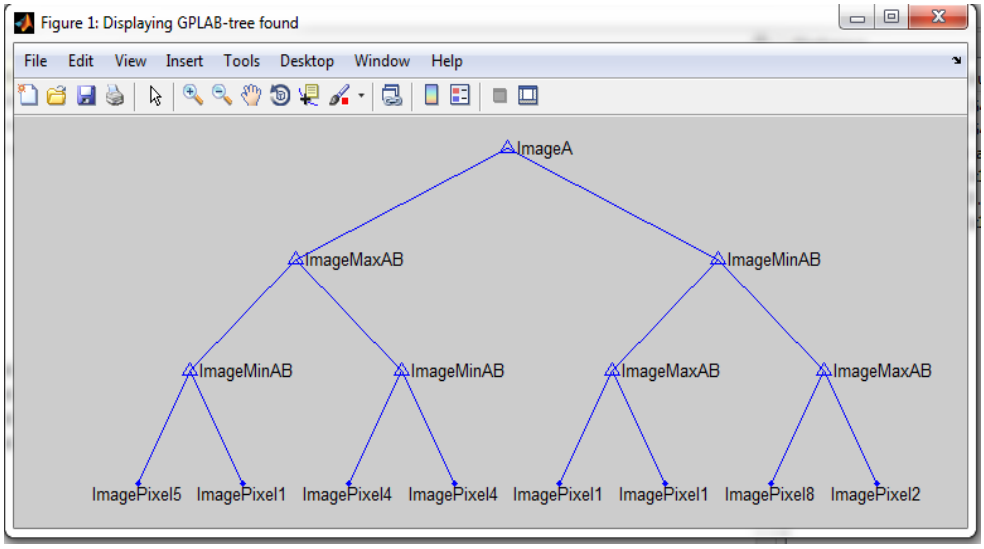

Figura 4.12 Exemplo de resultado obtido pela GPLAB.

O ruído Sal e Pimenta é adicionado as imagens com 5% dos *pixels* corrompidos, enquanto que o ruído Gaussiano é adicionado com média 0 e variância de 0,008. Nesse experimento, todos os filtros de imagem evoluídos são treinados usando-se a imagem “Cameraman”, em escala de cinza 8 *bits* por *pixel* e com 256×256 *pixels*.

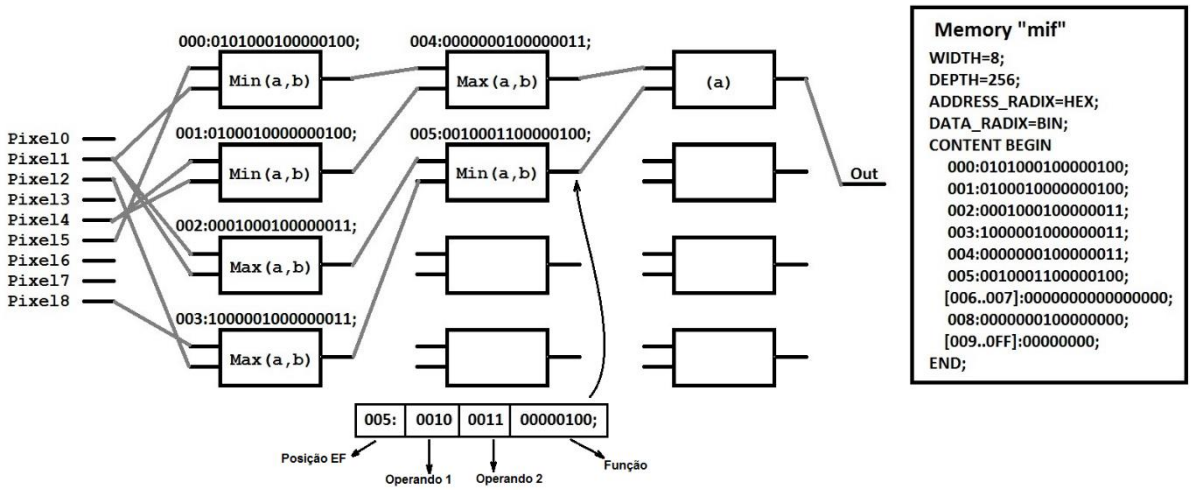

Figura 4.13 Modelo do fenótipo que se deseja implementar e arquivo de memória.

Capítulo 5

RESULTADOS

Neste capítulo, apresentam-se os resultados obtidos da arquitetura flexível e reconfigurável com simulações em: geração de circuitos lógicos e processamento digital de imagens. E faz-se uma comparação com outros trabalhos encontrados na literatura.

5.1 Resultados da Simulação da Geração de Circuitos Lógicos

Para este trabalho são efetuados testes para solucionar alguns circuitos lógicos digitais: somador (Add), um circuito aleatório (Random), ambos com 3 *bits* de entrada e uma saída, e a função XOR com 2 *bits* de entrada, utilizando o *software* Quartus II para a simulação, como também a família CycloneII (placa EP2C20F484C7), onde a matriz de EFs é configurada com 4 linhas e 3 colunas, totalizando-se 12 elementos com cromossomo de 96 *bits*.

Outras configurações da GPLAB foram testadas para se obter diferentes tamanhos de matrizes. A Tabela 5.1 traz os resultados obtidos da ocupação do *hardware* em diferentes tamanhos de matrizes de EF em relação à FPGA, que possui 18.752 elementos lógicos, 18.752 registradores lógicos e 239.616 *bits* de memória; a tabela apresenta quantas unidades de cada item são utilizadas e quanto representa de ocupação de *hardware* em cada matriz. Verifica-se que a ocupação do *hardware* depende apenas do tamanho da matriz que é necessária. Também, pode-se observar que existe pouca ocupação de *hardware,* principalmente da memória.

Na Figura 5.1, o gráfico de ocupação de *hardware* demonstra o aumento da ocupação em relação ao tamanho da matriz.

Tabela 5.1 – Custos de *hardware.*

FPGA	4X3	5X4	6X5	7X6
Elementos lógicos (18752)	160 (1%)	317 (2%)	499 (3%)	745 (4%)
Registradores lógicos (18752)	17 <1%	17 <1%	17 <1%	17 <1%
Bits de memória (239616)	512 <1%	512 <1%	512 <1%	512 <1%

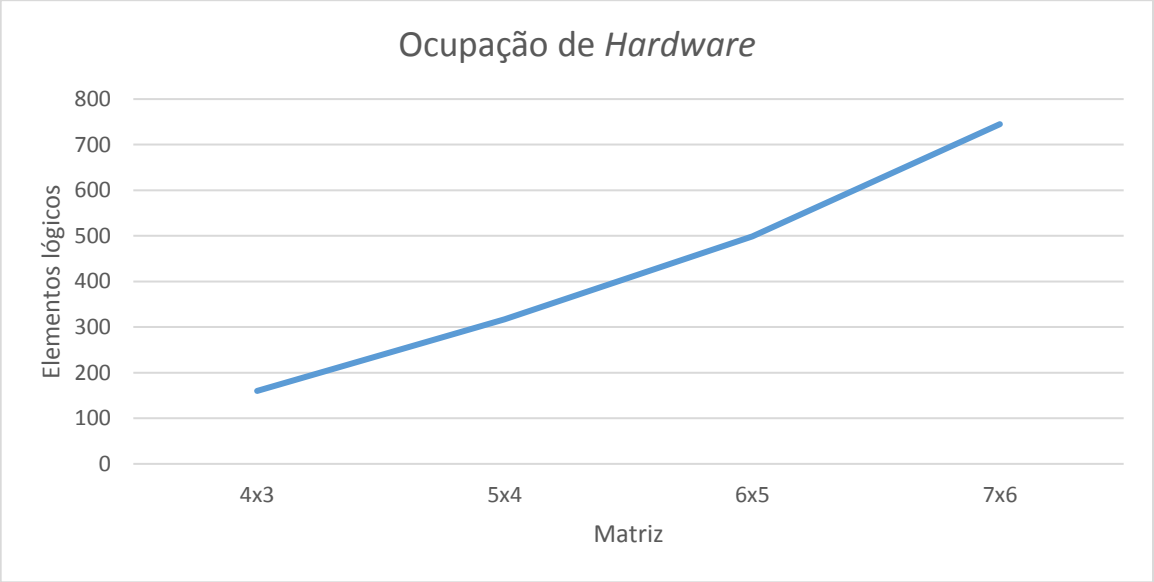

Figura 5.1 Ocupação de *hardware* por matriz.

Os resultados obtidos das simulações neste trabalho, para os circuitos lógicos descritos, apesentam uma resolução de forma rápida, como é demonstrado pelo gráfico de Ocorrência de Solução por Geração, apresentado na Figura 5.2; este demonstra a quantidade de ocorrências em cada geração, nas 30 execuções da simulação, onde se observa que a ferramenta GPLAB encontra uma solução ótima até a quarta geração de indivíduos, para os casos abordados.

Os resultados obtidos para os circuitos lógicos descritos são simulados utilizando o *software* Quartus II, e apresentam um atraso na saída menor que 20ns, nos três casos abordados. Este atraso é levemente melhor que o encontrado pelo trabalho de (VASU, SRIVASTAVA, *et al.*, 2014) que obtiveram um atraso de 22,828ns.

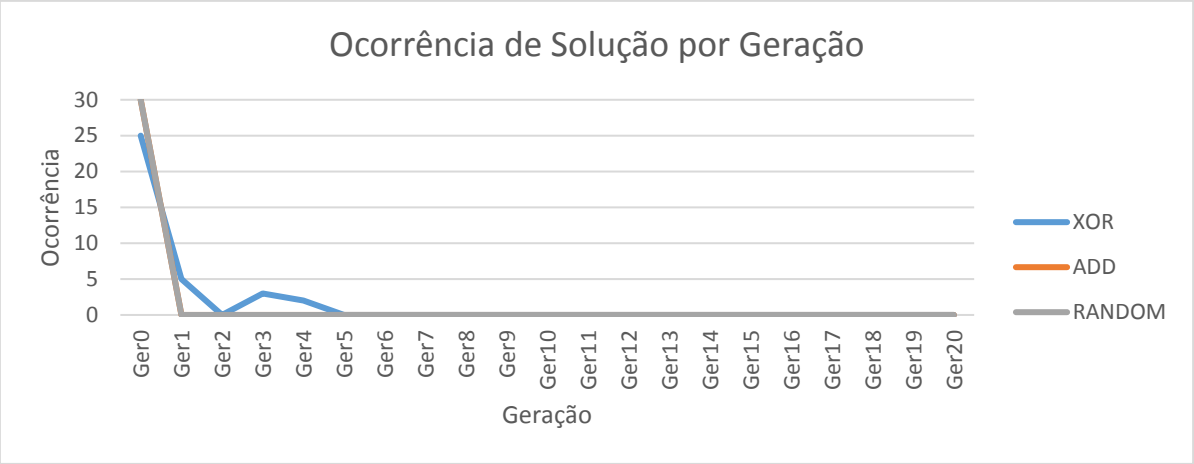

Figura 5.2 Ocorrência de Solução circuito XOR, RANDOM e ADD.

5.2 Resultados da Simulação da Aplicação de Filtros em Processamento Digital de Imagens

Para este trabalho foram efetuados testes para filtrar imagens com ruído adicionado, do tipo: Sal e Pimenta e Gaussiano, utilizando o *software* Quartus II para a simulação de *hardware*, em FPGA da família Cyclone II (EP2C20F484C7) da Altera, onde uma VRA foi elaborada com matriz de EFs de 4 linhas e 3 colunas, totalizando 12 elementos e cromossomo de 192 *bits*.

O ruído Gaussiano com média 0 e variância 0,008 e o ruído Sal e Pimenta com 5% de *pixels* corrompidos são adicionados às imagens Cameraman e Lena, possuindo resolução de 256×256 *pixels* em escala de cinza, e 8 *bits* por *pixel*. Ambos os ruídos são gerados usando funções do MATLAB. Dois tipos de filtros de imagem são evoluídos na GPLAB para processar cada um desses ruídos. Nesse experimento, todos os filtros de imagem são evoluídos e treinados usando a imagem Cameraman, em escala de cinza (8 *bits* por *pixel*) de 256×256 *pixels*.

Foram realizadas 100 evoluções independentes para cada configuração experimental e coletado o melhor MDPP, comparando-se as imagens filtradas com a imagem original. A função MDPP mede a média das diferenças entre o *pixel* da imagem filtrada e original. Quanto menor o valor da MDPP, melhor é a qualidade da

imagem filtrada. Os resultados obtidos das simulações para filtrar a imagem de entrada estão descritos conforme segue.

A ferramenta GPLAB apresentou as melhores soluções até a vigésima sétima geração, como demonstram as ocorrências de solução para as simulações de filtro para ruído Sal e Pimenta e ruído Gaussiano, o que demonstra a eficiência da ferramenta, da forma como foi configurada, conforme demonstram as Figuras 5.3 e 5.4. Nesses gráficos, o número total de ocorrências de solução a cada geração é demonstrado, para um total de 100 execuções da simulação.

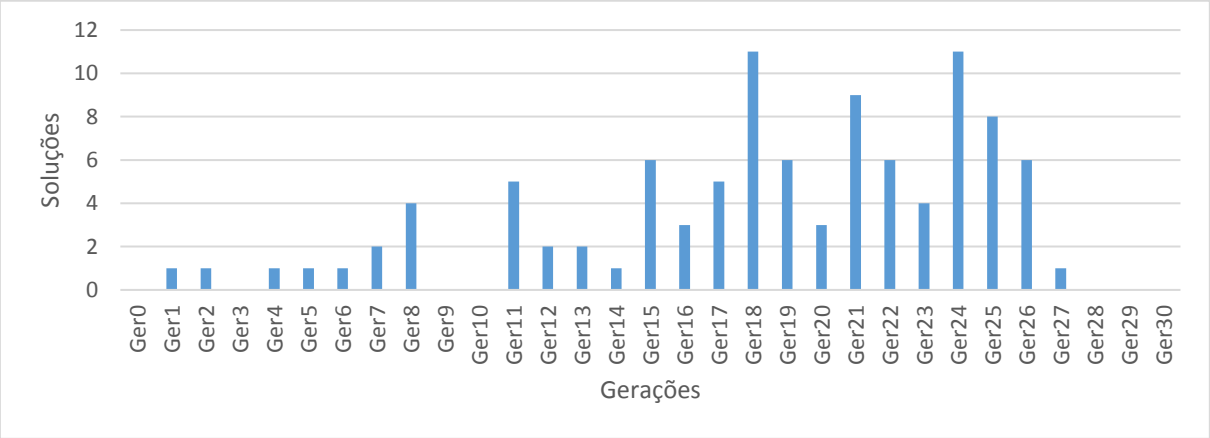

Figura 5.3 Ocorrência de solução na simulação do filtro Sal e Pimenta.

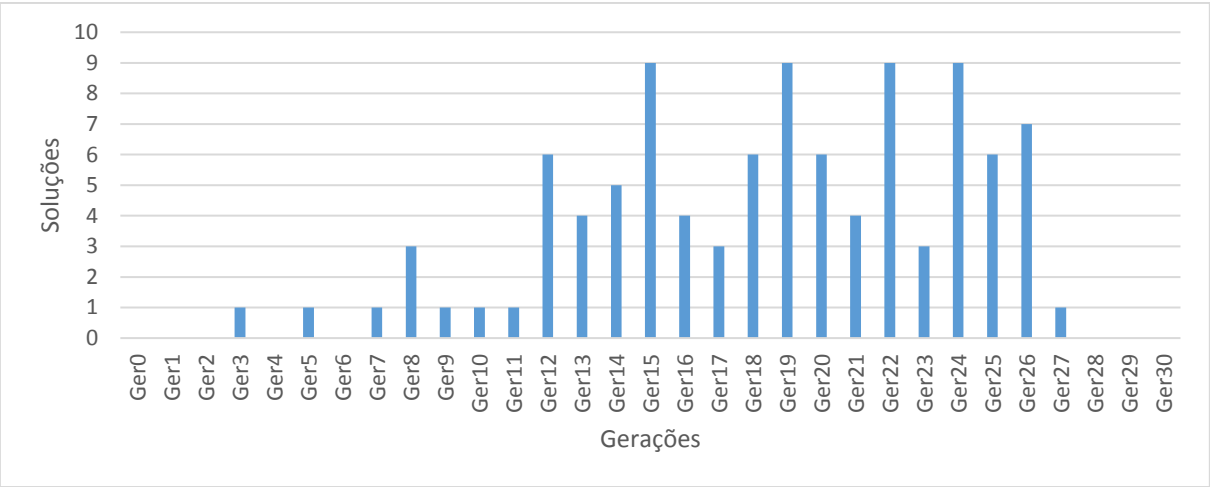

Figura 5.4 Ocorrência de solução na simulação do filtro Gaussiano.

Outras configurações da GPLAB foram testadas para obter diferentes tamanhos de matrizes aplicadas ao filtro de imagem. A Tabela 5.2 traz os resultados obtidos da ocupação do *hardware* em diferentes tamanhos de matrizes de EF, em relação à FPGA, que possui 18.752 elementos lógicos, 18.752 registradores lógicos e

239.616 *bits* de memória; a tabela apresenta quantas unidades de cada item são utilizados e quanto é a ocupação do *hardware* em cada matriz. Verifica-se que a ocupação do *hardware* depende apenas do tamanho da matriz que é necessária; também, existe pouca ocupação de *hardware,* principalmente da memória, que é muito pouco utilizada.

Tabela 5.2 – Custos de *hardware.*

FPGA	4X3	5X4	6X5	7X6
Elementos lógicos (18752)	1633 (9%)	2780(15%)	4312(23%)	6169(33%)
Registradores lógicos (18752)	167<1%	167<1%	167<1%	167<1%
Bits de memória (239616)	1632<1%	1632<1%	1632<1%	1632<1%

Na Figura 5.5, o gráfico de ocupação de *hardware* demonstra o aumento da ocupação em relação ao tamanho da matriz.

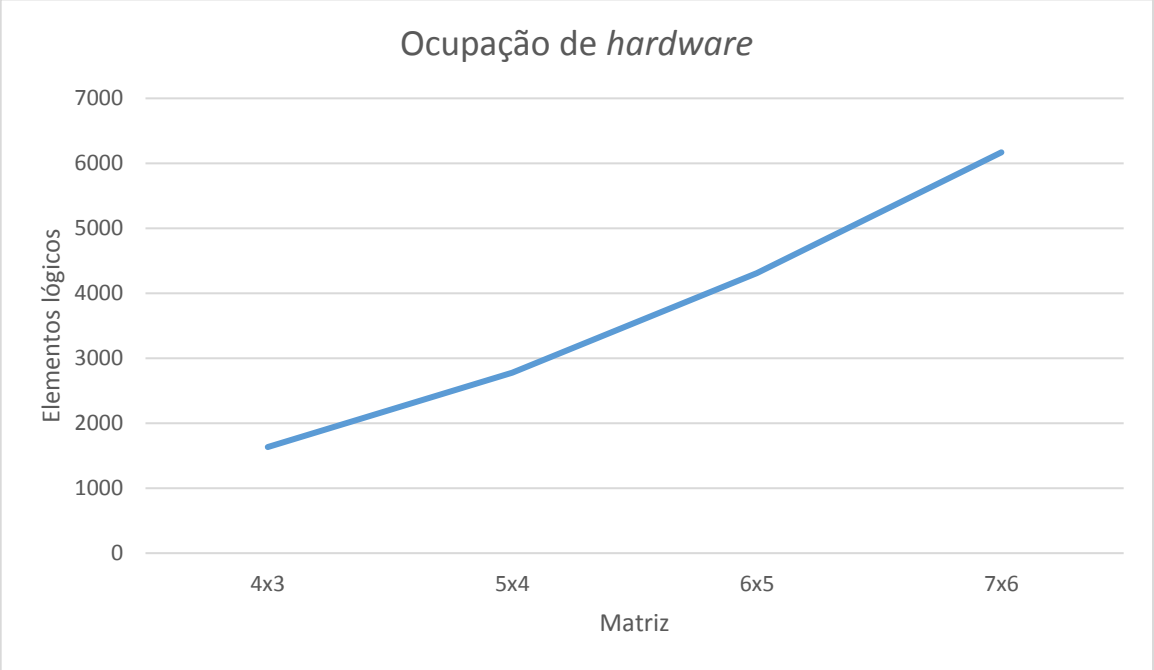

Figura 5.5 Ocupação de *hardware* por matriz.

Pode-se comparar os filtros obtidos com os resultados apresentados em outros trabalhos da literatura. Wang, Chen e Lee empregaram um *hardware* inspirado em PGC para filtragem de imagens para processar os mesmos ruídos, Sal e Pimenta e Gaussiano, nos quais obtiveram bons resultados; na Tabela 5.3 estão apresentados os resultados obtidos por essa literatura para o melhor MDPP, e treinados utilizando a imagem Lena. Os melhores indivíduos apresentados nessa literatura foram

simulados em MATLAB, porém, apresentaram resultados divergentes dos valores apresentados por essa literatura. A Tabela 5.4 demonstra esses resultados quando os filtros encontrados foram aplicados às imagens Lena e Cameraman.

Tabela 5.3 – Resultados de Wang, Chen e Lee em PGC treinados com a imagem Lena.

Ruído	Melhor MDPP	Média MDPP
Sal e Pimenta	1,54	2,30
Gaussiano	8,39	8,94

Tabela 5.4 – MDPP encontrado, por MATLAB, do melhor indivíduo apresentado por Wang, Chen e Lee.

Ruído	Lena	Cameraman
Sal e Pimenta	1,9140	5,5428
Gaussiano	20,2850	25,9728

A Tabela 5.5 demonstra que o desempenho dos filtros deste trabalho, com uma evolução satisfatória para as imagens de Lena e Cameraman, nos ruídos Sal e Pimenta e Gaussiano. Comparando-se com os valores encontrados em simulações dos filtros descritos na literatura, pode-se notar que para o ruído Gaussiano foram obtidos melhores valores de MDPP e para o ruído Sal e Pimenta foram obtidos valores próximos.

Tabela 5.5 – Resultados da Proposta.

	Cameraman		Lena	
Ruído	Melhor MDPP	Média MDPP	Melhor MDPP	Média MDPP
Sal e Pimenta	6,9665	8,8850	2,4469	4,2254
Gaussiano	12,7097	14,1719	12,3826	14,0422

Os resultados práticos, ao aplicar os filtros encontrados pela ferramenta GPLAB, são apresentados nas Figuras 5.6 e 5.7, e demonstram o bom desempenho dos filtros da Tabela 4.5.

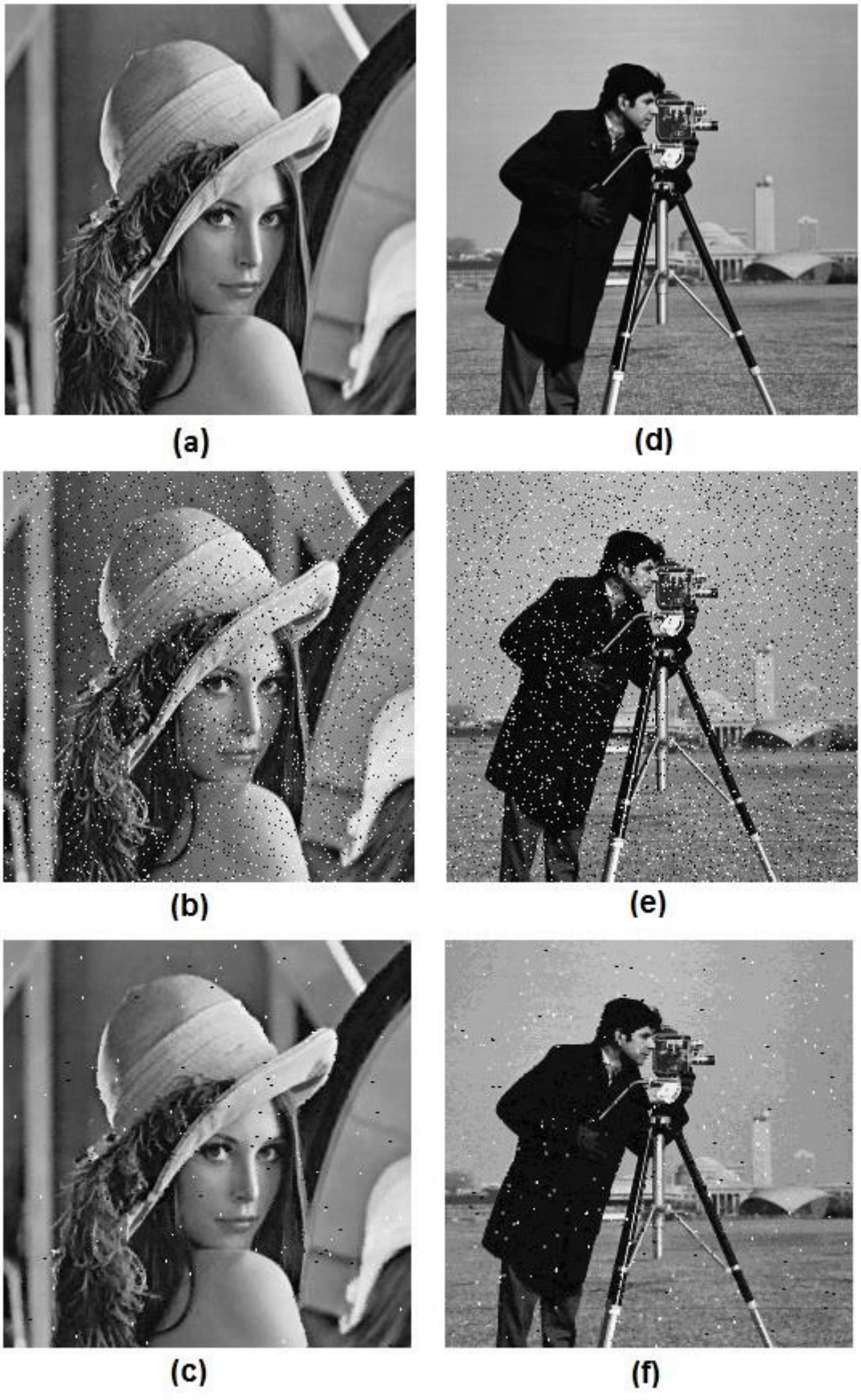

Figura 5.6 (a) Lena original, (b) com ruído e (c) filtrada MDPP=2,4469, (d) Cameraman original, (e) com ruído e (f) filtrada MDPP=6,9665, ruído Sal e Pimenta.

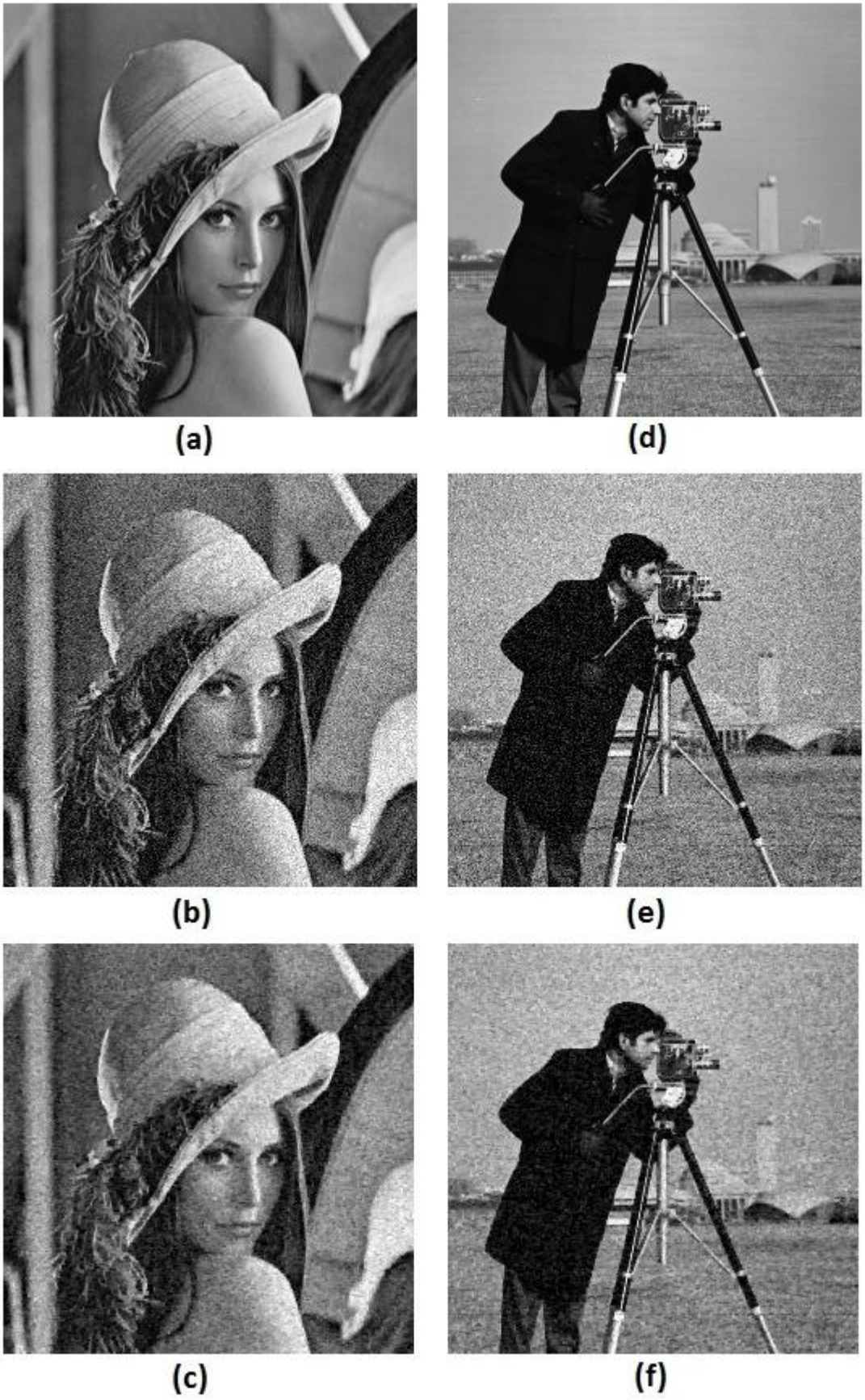

Figura 5.7 (a) Lena original, (c) com ruído e (e) filtrada MDPP=12,8123, (b) Cameraman original, (d) com ruído e (f) filtrada MDPP=12,7097, ruído Gaussiano.

Os *hardwares* obtidos para os filtros de imagem descritos foram simulados utilizando o *software* QuartusII e uma imagem de teste simplificada de 10×10 *pixels*, tornando possível acompanhar o processamento *pixel* a *pixel*. Essa imagem foi obtida por meio de funções do MATLAB, extraindo-se uma fração da imagem Cameraman. Na Tabela 5.6, tem-se os valores dessa fração de imagem *pixel* a *pixel*. Na Tabela 5.7 estão os valores obtidos após a aplicação do filtro em MATLAB. Esses resultados foram usados para verificar os valores que foram efetivamente obtidos pelo *hardware*.

Tabela 5.6 – Valores de pixel Imagem 10X10 com ruído.

153	170	153	153	153	153	170	153	153	170
153	153	153	170	153	153	153	153	170	153
153	170	153	153	170	153	170	153	153	153
153	153	255	153	153	153	153	170	153	170
153	153	153	153	170	153	153	153	153	153
153	153	170	153	153	170	153	153	153	153
153	153	153	153	153	153	153	153	153	170
170	153	153	170	153	153	153	170	153	153
153	153	153	153	153	153	170	153	153	170
153	153	170	153	170	153	170	153	170	153

Tabela 5.7 – Valores de pixel Imagem filtrada.

153	153	153	153	153	153	153	170
153	153	153	153	153	153	153	170
153	153	153	153	153	153	153	153
153	153	153	153	153	153	153	153
153	153	153	153	153	153	153	153
153	153	153	153	153	153	153	153
153	153	153	153	153	153	153	153
153	153	170	153	153	170	170	153

A função de filtro obtida e utilizada nessa simulação foi selecionada aleatoriamente do conjunto de 100 soluções encontradas pela ferramenta GPLAB, para elaborar o filtro de Sal e Pimenta, e está apresentada na Figura 5.8.

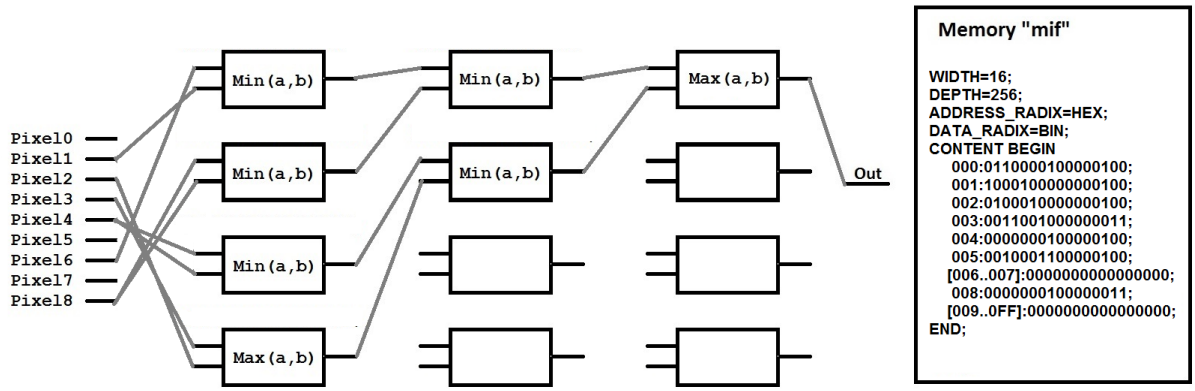

Figura 5.8 Descrição do *hardware* da solução encontrada filtro Sal e Pimenta.

Esta simulação apresenta um atraso entre o início da leitura da imagem em relação ao início da saída de *pixels* tratados, em função do tamanho desta imagem para construir o registrador FIFO necessário, como demonstrado na equação 11; multiplicando o resultado dessa equação pelo atraso de processamento de cada *pixel* que é da ordem de 40 ns, obtém-se, então, um atraso de 920ns após o sinal de *reset* ser inibido.

$$T = 2 * 10 + 3 \tag{11}$$

Sendo a imagem de 256×256 *pixels,* o atraso de processamento será de 515 * 40 ns = 20,6 us, e o tempo de processamento será 256 * 256 * 40ns = 2,622 ms, portanto, capaz de processar 380 quadros por segundo nessa resolução. Para uma imagem de resolução de 1024×1024 *pixels,* o tempo de processamento será de 42 ms, e o circuito será capaz de processar 23,8 quadros por segundo.

Capítulo 6

CONCLUSÃO

Neste capítulo de conclusão é apresentada uma síntese dos principais resultados e da proposta para solução do problema estabelecido inicialmente.

6.1 Considerações Finais

Neste trabalho, é proposto um *hardware* evolutivo, utilizando uma arquitetura virtual reconfigurável, na qual a evolução ocorre em uma ferramenta de Programação Genética. Nesta proposta, o melhor indivíduo é evoluído e passado ao *hardware* para a sua reconfiguração. O sistema foi implementado utilizando a ferramenta GPLAB para MATLAB. Esta, executa as evoluções através de PG e o resultado é convertido para *hardware*, por meio de rotinas em MATLAB e do *software* Quartus II da Altera. O sistema foi aplicado na solução de circuitos lógicos e na filtragem de ruídos de imagens.

6.2 Contribuições e Limitações

Pode-se notar, por meio dos resultados obtidos, que a ferramenta GPLAB é um ótimo recurso para o estudo da programação genética e que esse recurso pode ser utilizado para gerar resultados no formato cartesiano, no qual este pode ser implementado em um sistema embarcado que utiliza uma AVR.

Este trabalho demonstra que a GPLAB é uma ferramenta adequada para gerar soluções de forma eficiente. Demonstra-se também como a AVR Hibrida pode ser usada como um recurso de avaliação e implementação de *hardware* em conjunto com a ferramenta GPLAB, para obterem-se soluções eficientes de *hardware*.

As simulações das soluções obtidas demonstraram os seguintes benefícios:

- Pequenos atrasos para o *hardware* de circuitos lógicos, na ordem de 20 ns e ligeiramente melhor que o encontrado na literatura.
- Atraso no processamento de filtros de imagem da ordem de 40 ns, o que permite aplicar o sistema em uma aplicação de aquisição de vídeo.
- Pouca ocupação de *hardware* e ocupação com crescimento proporcional ao crescimento da matriz de EF.
- Pouca ocupação de memória da ordem de menos que 1% do disponível na FPGA.

O uso da GPLAB não elabora resultados com múltiplas saídas, o que dificulta a criação de circuitos lógicos mais sofisticados.

A AVR Hibrida utilizando FPGA da família Cyclone II (EP2C20F484C7), e simulada no Quartus II, apresenta uma limitação na leitura de memória da FPGA que não permite fazer leituras mais rápidas que 20 ns, limitando-se o processamento de imagens a uma resolução de 1024×1024 *pixels*, em tons de cinza e 8 *bits* por *pixel*, a 24 quadros por segundo.

6.3 Lições aprendidas

A ferramenta GPLAB apresenta uma boa alternativa para o estudo de PG, o que facilita o aprendizado de PG, tornando as simulações mais simples e mais rápidas. É uma ferramenta que pode ser amplamente utilizada, apesar de não ter sido completada e algumas opções de utilização ainda não estarem disponíveis.

Arquitetura Virtual Reconfigurável é uma solução eficiente na elaboração de *hardware* evolucionário, a qual pode utilizar um cromossomo de configuração de forma otimizada, utilizando pouco *hardware*.

6.4 Trabalhos Futuros

Pretende-se implementar um algoritmo mais eficiente de PGC, para ser utilizado no lugar do GPLAB, que utilize a VRA, em uma tecnologia de FPGA, utilizando-se para isso técnicas de processamento dedicado em FPGA.

REFERÊNCIAS

ALTERA CORPORATION. Cyclone II Architeture. In: _____ **Cyclone II device handbook**. [S.l.]: [s.n.], v. 1, 2007. Disponivel em: <http://www.altera.com/literature/hb/cyc2/cyc2_cii51002.pdf>. Acesso em: set. 2014.

BARTOLINI, D. B. et al. **HERA:** Hardware evolution over reconfigurable architectures. Computing in Heterogeneous, Autonomous 'N' Goal-Oriented Environments. [S.l.]: [s.n.]. 2011. p. 1-8.

CANCARE, F. et al. **A Bird's Eye View of FPGA-based Evolvable Hardware**. NASA/ESA Conference on Adaptive Hardware and Systems. [S.l.]: [s.n.]. 2011.

CANCARE, F. et al. **DGECS:** Description Generator for Evolved Circuits Synthesis. Parallel and Distributed Processing Symposium Workshops & PhD Forum. [S.l.]: [s.n.]. 2012. p. 454-461.

CANCARE, F.; SANTAMBROGIO, M.; SCIUTO, D. **A Direct Bitstream Manipulation Approach for Virtex4-based Evolvable Systems**. Proceedings of thee IEEE International Symposium on Circuits and Systems. [S.l.]: [s.n.]. 2010. p. 853-856.

DARWIN, C. **A Origem das Espécies por Meio da Seleção Natural**. Londres: [s.n.], 1859.

EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computing**. Berlim: Springer, 2003.

FOGEL, L. J. Autonomous Automata. **Industrial Research Magazine**, v. 4, p. 14-19, Fevereiro 1962.

GABRIEL, P. H. R.; DELBEM, A. C. B. **Fundamentos de algoritmos evolutivos**. São Carlos: ICMC-USP, 2014. Disponivel em: <http://www.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_75.pdf>. Acesso em: ago. 2014.

GALLAGHER, J. C.; VIAGRAHAM, S. **A modified compact genetic algorithm for the intrinsic evolution of continuous time recurrent neural networks**. Proceedings of the Genetic and Evolutionary Computation Conference, ser. GECCO ’02. San Francisco: Morgan Kaufmann Publishers Inc. 2002. p. 163-170.

GALVÃO, C. O.; VALENÇA, M. J. S. **Sistemas inteligentes:** aplicações a recursos hidricos e ciencias ambientais. Porto Alegre: Ed. Universidade/UFRGS, 1999.

GLETTE, K.; KAUFMANN, P. **Lookup Table Partial Reconfiguration for an Evolvable Hardware Classifier System**. 2014 IEEE Congress on Evolutionary Computation. [S.l.]: [s.n.]. 2014. p. 1706-1713.

GLETTE, K.; TORRESEN, J.; HOVIN, M. **Intermediate level fpga reconfiguration for an online ehw pattern recognition system**. Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference. [S.l.]: [s.n.]. 2009. p. 19-26.

GLETTE, K.; TORRESEN, J.; YASUNAGA, M. **On-chip evolution using a soft processor core applied to image recognition**. Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference. [S.l.]: [s.n.]. 2006. p. 373-380.

GOEL, A. PLDs. **slideshare**, 10 jan. 2016. Disponivel em: <http://pt.slideshare.net/anishgoel/plds>. Acesso em: 2016.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 2. ed. New Jersey: Prentice Hall, 2001.

HOLLAND, J. H. Outline for a Logical Theory of Adaptative Systems. **Journal of the ACM**, New York, v. 9, n. 3, p. 297-314, jul. 1962.

HUELSBERGEN, L.; RIETMAN, E. Evolving oscillators in silico. **Evolutionary Computation, IEEE Transactions on, vol. 3, no. 3**, 1999. 197-204.

JIANAN, L. et al. **Digital Circuit Network Evolution Based on the Virtual Reconfigurable Platform**. Computational Intelligence and Communication Networks. [S.l.]: [s.n.]. 2012. p. 540-544.

JONG, K. A. D. **Evolutionary computation**. London: The MIT Press, 2008.

KALGANOVA, T. **Bidirectional Incremental Evolution in Extrinsic Evolvable Hardware**. Proc. of the European Conference on Genetic Programming, EuroGP2000. Edinburgh: [s.n.]. 2000. p. 60-75.

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

LAMBERT, C.; KALGANOVA, T.; STOMEO, E. FPGA-based Systems for Evolvable Hardware. **International Journal of Electrical, Computer, and Systems Engineering**, v. 3, p. 62-68, 2009.

LEVI, D.; GUCCIONE, S. A. **Geneticfpga:** Evolving stable circuits on mainstream fpga devices. Washington DC: [s.n.]. 1999.

MARQUES, O.; VIEIRA, H. **Processamento Digital de Imagens**. Rio de Janeiro: Brasport, 1999.

MATHWORKS. Mathworks Accelerating the pace of engineering and science. **Mathworks Accelerating the pace of engineering and science**, 10 jul. 2014. Disponivel em: <http://www.mathworks.com/>. Acesso em: 10 jul. 2014.

MILLER, J. F. An empirical study of the efficiency of learning boolean functions using. **GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE**, Edinburgh, p. 1135-1142, 1999. Disponivel em: <http://www.cartesiangp.co.uk/papers/gecco1999b-miller.pdf>. Acesso em: jun. 2014.

MILLER, J. F. **Cartesian Genetic Programming**. Berlim: Springer-Verlag, 2011. 365 p.

MILLER, J. F.; THOMSON, P.; FOGARTY, T. C. Designing electronic circuits using. **Genetic Algorithms and Evolution Strategies in Engineering and Computer Science**, Chichester, 1997.

OREIFEJ, R. et al. **Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of virtex ii pro devices**. Field Programmable Logic and Applications, 2007. FPL 2007. International Conference. [S.l.]: [s.n.]. 2007. p. 299-304.

PEDRINO, E. C. **Arquitetura Pipeline Reconfiguravel Através de Instruções Geradas por Programação Genetica**. São Carlos: Universidade de Engenharia de São Carlos, 2008. 220 p. Tese Doutorado Processamento de sinais e instrumentação.

PEDRINO, E. C. et al. **INTELLIGENT FPGA BASED SYSTEM FOR SHAPE RECOGNITION**. Southern Programmable Logic (SPL). Córdoba: [s.n.]. 2011.

PEDRINO, E. C.; OGASHAWARA, O.; RODA, V. O. **RECONFIGURABLE ARCHITECTURE FOR MATHEMATICAL MORPHOLOGY USING GENETIC PROGRAMMING AND FPGAS**. Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. Atlanta: IEEE. 2010. p. 1-4.

PORTER, R. B.; BERGMANN, N. W. **Evolving fpga based cellular automata**. Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning. London: Springer-Verlag. 1999. p. 114-121.

RAYWARD-SMITH, V. J. et al. **Modern Heuristic Search Methods**. Chinchester: Wiley, 1996.

RECHENBERG, I. **Cybernetic Solution Path of an Experimental Problem**. [S.l.]: Royal Aircraft Establishment Library Translation, 1964.

REEVES, C. R. **Modern Heuristic Techniques for Combinatorial Problems**. New York: Wiley, 1993.

REN, A. et al. **Implement of evolable hardware based improved genetic algorithm**. Natural Computation (ICNC), 2011 Seventh International Conference. [S.l.]: [s.n.]. 2011. p. 2112-2115.

SCHWEFEL, P. H. **Numerical Optimization of Computer Models**. New York: John Wiley & Sons, Inc, 1981.

SCOPUS. scopus. **www.scopus.com/**, 10 jul. 2014. Disponivel em: <http://www.scopus.com/>.

SEKANINA, L. Evolutionary functional recovery in virtual reconfigurable circuits. **J. Emerg. Technol. Comput. Syst.**, v. 3, 2007.

SEKANINA, L.; FRIEDL, S. An evolvable combinational unit for fpgas. **Computing and Informatics**, v. 23, p. 461-486, 2004.

SILVA, S. **GPLAB:** a genetic programming toolbox for MATLAB. [S.l.]: [s.n.], 2007. Disponivel em: <http://klobouk.fsv.cvut.cz/~leps/teaching/mmo/data/gplab.manual.3.pdf>. Acesso em: fev. 2014.

SRIVASTAVA, A. K. et al. **Design and simulation of virtual reconfigurable circuit for a Fault Tolerant System**. Recent Advances and Innovations in Engineering. [S.l.]: [s.n.]. 2014. p. 1-4.

STOMEO, E.; KALGANOVA, T.; LAMBERT, C. Generalized disjunction decomposition for evolvable hardware. **Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 36, no. 5**, 2006. 1024-1043.

THOMPSON, A. **An evolved circuit, intrinsic in silicon, entwined with physics**. Proceedings of the First International Conference on Evolvable Systems: From Biogogy to Hardware. London: Springer-Verlag. 1996. p. 390-405.

THOMPSON, A. **On the automatic design of robust electronics through artificial evolution**. Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware. London: Springer-Verlag. 1998. p. 13-24.

TORRESEN, J. **Increased complexity evolution applied to evolvable hardware**. Smart Engineering System Design: Neural Networks, Fuzzy Logic , Evolutionary Programming, Data Mining, and Complex Systems Proc. of ANNIE'99'. [S.l.]: [s.n.]. 1999. p. 429-436.

TYRRELL, A.; HOLLINGWORTH, G.; SMITH, S. **Evolutionary strategies and intrinsic fault tolerance**. Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop. [S.l.]: [s.n.]. 2001. p. 98-106.

UPEGUI, A.; SANCHEZ, E. Evolving hardware by dynamically reconfiguring xilinx fpgas. **Lecture Notes in Computer Science**, v. 3637, p. 56-65, 2005.

UPEGUI, A.; SANCHEZ, E. **Evolving hardware with self-reconfigurable connectivity in xilinx fpgas**. Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems. Washington, DC: [s.n.]. 2006. p. 153-162.

VAHID, F.; GIVARGIS, T. **Embedded system design:** a unified hardware/software approach. Riverside: Departament of Computer Science and Engineering University of California, 1999.

VASICEK, Z.; SEKANINA, L. **Evaluation of a new platform for image filter evolution**. Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference. [S.l.]: [s.n.]. 2007. p. 577-586.

VASICEK, Z.; SEKANINA, L.; BIDLO, M. **A method for design of impulse bursts noise filters optimized for fpga implementations**. Design, Automation Test in Europe Conference Exhibition (DATE), 2010. [S.l.]: [s.n.]. 2010. p. 1731-1736.

VASU, R. et al. **Design and Simulation of Virtual Reconfigurable Circuit for a Fault Tolerant System**. IEEE International Conference on Recent Advances and Innovations in Engineering (JCRAIE-2014). Jaipur India: [s.n.]. 2014. p. 1-4.

WANG, J. R.; WANG, D.; LAI, J. M. **A hierarchical parallel evolvable hardware based on network on chip**. International Conference on Reconfigurable Computing and FPGAs. [S.l.]: [s.n.]. 2013. p. 1-6.

WANG, J.; CHEN, Q. S.; LEE, C. H. Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. **Published in IET Computers & Digital Techniques**, v. 2, p. 386-400, Dezembro 2007.

WANG, J.; PIAO, C. H.; LEE, C. H. **Implementing multi-vrc cores to evolve combinational logic circuits in parallel**. Proceedings of the 7th international conference on Evolvable systems: from biology to hardware. Berlim: Springer-Verlag. 2007. p. 23-34.

XPLORE, IEEE. IEEE xplore digital library. **IEEE xplore digital library**, 10 jul. 2014. Disponivel em: <http://ieeexplore.ieee.org/Xplore/home.jsp>. Acesso em: jun. 2014.

YASUNAGA, M.; YOSHIHARA, I.; KIM, J. H. **Gene finding using evolvable reasoning hardware**. Proceedings of the 5th international conference on Evolvable systems: from biology to hardware. Berlim: [s.n.]. 2003. p. 198-207.