TEMPLATES DE ALGORITMOS GENÉTICOS PARA A GERAÇÃO DE ESTÍMULOS APLICADOS À VERIFICAÇÃO FUNCIONAL DE DISPOSITIVOS

Ricardo A. P. Franco, Karina R. G. da Silva
Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação, Goiânia – Goiás

***Resumo* - Com o rápido crescimento na complexidade de circuitos eletrônicos modernos, houve a necessidade do desenvolvimento de novas ferramentas e metodologias para otimizar o processo de criação de um *Intellectual Property Core*. Uma fase fundamental nesse processo é a fase de verificação. A verificação funcional é responsável por garantir que as funcionalidades de um dispositivo recém desenvolvido, foram implementadas corretamente. A metodologia VeriSC foi desenvolvida para otimizar a fase de verificação funcional, gerando automaticamente *templates*, para o ambiente de testes, diminuindo assim o tempo total de projeto. O objetivo deste artigo é criar uma base para a criação de algoritmos genéticos para a geração de dados. Esses algoritmos são aplicados na metodologia VeriSC. A geração de dados será padronizada em um *template* que insere essa nova abordagem na metodologia VeriSC. Os resultados obtidos demonstram quais etapas do algoritmo genético foram padronizadas e quais não são possíveis, pois são dependentes diretamente do problema a ser resolvido. O artigo é concluído analisando a contribuição dessa nova abordagem de geração de dados e os benefícios para a equipe de verificação.**

***Palavras-Chave* - Algoritmo genético, geração de dados e verificação funcional.**

A STUDY OF GENETIC ALGORITHMS APPLIED TO FUNCTIONAL VERIFICATION

***Abstract* - With the fast growth in complexity of modern electronic circuits, there was a need to development of new tools and methodologies to optimize the process of creating an Intellectual Property Core. The key phase in this process is the verification phase. The functional verification is responsible for ensuring that the functionality of a newly developed device were implemented correctly. The VeriSC methodology was developed to optimize the functional verification phase automatically generating templates of testbench, that decreases the design time. The goal of this paper is to create a basis for the creation of genetic algorithms for data generation. These algorithms are applied in VeriSC methodology. The data generation will be patterned in a template that inserts this new approach in VeriSC methodology. The results show which steps of genetic algorithm were patterned and which are not possible because they are directly dependent on the problem to be solved. The paper concludes by analyzing the contributions of this new approach of data generation and the benefits for the verification team.**

***Keywords* - *Functional verification, data generation and genetic algorithm*.**

I. INTRODUÇÃO

A criação de um *Intellectual Property Core* (IP *Core*) deve seguir fases preestabelecidas para que o componente de *hardware* a ser desenvolvido obtenha a funcionalidade desejada. Dentre essas fases, a fase de verificação é de suma importância para a validação das funcionalidades especificadas do dispositivo. Essa importância é evidenciada observando dois pontos cruciais:

- Nesta etapa é consumida cerca de 70% de todos os recursos de projeto [1];
- A etapa de verificação consome aproximadamente 50% de todo o tempo de projeto [2].

A verificação é um processo usado para demonstrar que o objetivo de um projeto é preservado em sua implementação [1]. A verificação funcional baseia-se em aplicar estímulos em dois modelos idênticos implementados, um modelo ideal (descrito em alto nível) e um real (descrito em nível de sinais). A saída desses modelos, isto é, a resposta aos estímulos aplicados, são comparadas e elas deverão ser idênticas [1].

O conjunto de estímulos que contém todos os dados necessários para realizar a verificação funcional é chamado de cobertura funcional [2]. No momento em que esses estímulos forem aplicados em ambos os modelos e suas respostas forem idênticas, diz-se que a cobertura foi atingida, ou seja, todos os estímulos necessários para comprovar as funcionalidades do dispositivos foram aplicados e todas as respostas geradas pelos modelos foram satisfatórias [2].

O trabalho [2] propõe a criação de uma ferramenta e da metodologia VeriSC que auxilia a reduzir o tempo total de verificação e encontrar erros rapidamente, durante o projeto de um *IP Core*.

Os AGs têm como base a Teoria da Evolução das Espécies desenvolvida por Charles Darwin descrita em seu livro *On the origin of species* [3-5].

XII CEEL – ISSN 2178-8308
13 a 17 de Outubro de 2014
Universidade Federal de Uberlândia – UFU
Uberlândia – Minas Gerais – Brasil

Esses algoritmos possuem um conjunto de soluções, que formam uma população, geradas aleatoriamente e impostas à pressões do ambiente que causam a seleção natural [3-4]. Uma função de aptidão é descrita para mapear o problema a ser resolvido para o AG. Essa função é responsável por avaliar as soluções atribuindo uma nota de aptidão, relacionando o quão bom essa solução é perante a resolução do problema. As soluções que obtiveram as melhores avaliações deverão ser selecionadas para gerar uma nova população. Os operadores genéticos - recombinação e mutação - são aplicados, modificando as soluções selecionadas para gerar novas soluções, que deverão ser melhores que suas precursoras [3-5].

Sob esta ótica, o conceito de AG se torna uma ferramenta importante, pois são métodos que simulam os processos de evolução natural para, principalmente, resolver problemas de otimização [6].

O trabalho [7] propõe a verificação do comportamento de um *software under verification* por meio de um AG realizando a verificação funcional. O AG desenvolvido para a cobertura guiada permitiu otimizar automaticamente todo o espaço de busca das possíveis variáveis de entrada, aumentando assim a eficiência dos testes durante a fase de verificação do *software*. O trabalho proposto por este artigo, por outro lado, realiza a verificação funcional de componentes de *hardware* utilizando AG. O AG guia a geração de dados permitindo uma verificação mais robusta.

Em [8], uma unidade de geração de testes foi desenvolvida para encontrar bons parâmetros para unidades de testes aleatórios, otimizando assim o teste de cobertura. Essa unidade possui dois níveis, alto nível e baixo nível. No alto nível é desenvolvido um AG para gerar bons dados e enviá-los ao dispositivo de baixo nível que testa um conjunto de métodos de uma unidade de motor de testes randômicos. Esse trabalho demonstrou a superioridade do uso dos AGs na otimização dos parâmetros e na redução do tempo de verificação, além de realizar a comunicação entre o baixo nível e o alto nível.

A. Samarah, em [9], desenvolve um algoritmo chamado de Cell-based Genetic Algorithm (CGA) para otimizar a cobertura funcional. O AG é utilizado para aprimorar os parâmetros presentes nas células, na qual cada célula corresponde a uma representação de subconjuntos do domínio dos dados de entrada. Os resultados demonstraram que utilizando o algoritmo CGA, a verificação funcional foi otimizada.

No trabalho [10] é proposto a geração automatizada de *templates* do ambiente de teste (*testbench*) de *design under verification* (DUV) específicos. Esses *templates* são baseados na metodologia VeriSC, que é caracterizada por ser uma metodologia genérica e compreende todos os tipos de DUV síncronos. No entanto, a verificação implementada pela metodologia VeriSC utiliza dados pseudorrandômicos que geram em toda a execução os mesmos dados e capturam os mesmos erros. Este artigo proposto aborda um aprimoramento na geração dos dados, gerando-os de maneira aleatório-guiada e padronizando essa geração para facilitar a incorporação dela na metodologia VeriSC.

Pode-se notar, então, que a otimização da geração de dados na verificação funcional ainda é uma lacuna a ser resolvida, possibilitando o uso de ferramentas e metodologias que auxiliem durante essa fase.

O objetivo deste artigo é apresentar um *template*, que é uma padronização da implementação de algoritmos genéticos. Ele introduz uma nova abordagem na geração de dados da metodologia VeriSC. A nova geração de dados é realizada de maneira aleatório-guiada, ao invés de pseudorrandômica, utilizando os algoritmos genéticos. O *template* possui as funcionalidades dos AGs que são possíveis de padronização e as insere dentro da metodologia VeriSC. As partes não possíveis de serem reusadas são as partes dependentes das funcionalidades do problema a ser resolvido e devem ser feitas manualmente pelo engenheiro de verificação.

A utilização de estímulos guiados permitem uma verificação mais profunda, a otimização do tempo gasto na fase de verificação funcional e, consequentemente, a redução do gasto em recursos no projeto e rápido *time-to-market*.

II. ALGORITMOS GENÉTICOS

Muitos biólogos, nos anos de 1950 e 1960, desenvolveram simulações computacionais de sistemas genéricos, mas foi John Holland quem iniciou as primeiras pesquisas a respeito deste tema [11]. O ponto inicial dos AGs ocorreu quando J. Holland publicou o livro *Adaptation in Natural and Artificial Systems*, em 1975 [12].

Um algoritmo genético contém um conjunto de soluções - chamadas também de indivíduos - que formam uma população. Essa população será avaliada pela função de aptidão e parte de seus indivíduos serão selecionados para gerar novos indivíduos. A seleção visa encontrar, na população atual, as melhores soluções avaliadas pela função de aptidão. A combinação de boas soluções tendem a gerar soluções ainda melhores do que suas precursoras, caracterizando assim um processo evolutivo de soluções a cada nova geração.

Cada indivíduo corresponde a uma solução codificada para ser utilizada pelo AG. Os indivíduos serão avaliados pela função de aptidão para analisar o quão bom a solução é para o problema apresentado. Os indivíduos mais aptos terão mais chances de serem selecionados para participarem da formação da próxima geração.

As novas gerações serão obtidas através de operadores genéticos, pelos processos da reprodução e da mutação, aplicados sobre os indivíduos selecionados na etapa anterior. A reprodução mais simples ocorre com a seleção de dois indivíduos e a realização da troca de seus materiais genéticos por meio da recombinação de seus genes. Os indivíduos recém gerados poderão sofrer mutações em alguns de seus genes, alterando seus valores [3]. A figura 1 demonstra o funcionamento de um AG.

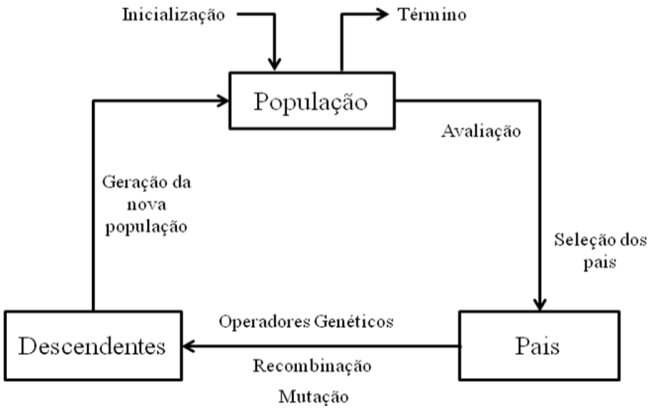

Fig. 1. Esquema simplificado de um Algoritmo Genético.

Esse algoritmo deverá ser repetido até que as soluções do algoritmo sejam satisfatórias, ou seja, até que os critérios de parada, determinados pelo programador, sejam atingidos.

III. METODOLOGIA VERISC

Sistemas computacionais podem ser constituídos por vários componentes de *hardware* que possuem funcionalidade específicas [2; 13]. A fase de verificação de *hardware* é crucial dentro do ciclo de desenvolvimento de um projeto de *hardware*.

A verificação de um dispositivo de hardware comprova se suas funcionalidades foram implementadas corretamente e se o dispositivo está livre de erros. Há três formas de realizar a verificação: verificação formal, funcional ou hibrida [2].

A verificação formal utiliza mecanismos de verificação estática e pode provar a inexistência de erros através de equações matemáticas e verificação de modelos. No entanto, esse processo pode ser complicado e podem ocorrer limitações do tamanho do circuito a ser verificado [2].

A verificação híbrida combina técnicas da verificação formal e funcional [2].

Na verificação funcional, a geração de dados é realizada de forma automatizada. Um gerador de dados produz estímulos válidos baseados nas restrições de um domínio particular [1]. A geração de estímulos é realizada até que toda a cobertura funcional seja atingida [2]. Técnicas e metodologias vêm sendo desenvolvidas para otimizar a geração de estímulos [7; 14; 15].

O processo de verificação é uma parte crítica do projeto de um *hardware* devido à escassez de projetos qualificados e de engenheiros de verificação [1]. Considerar, nesses projetos, a realização da verificação apenas após a conclusão do projeto agrava o problema [2]. Esses fatos justificam buscas por novas ferramentas e metodologias que permitam reduzir o tempo total de verificação.

A metodologia VeriSC realiza verificação funcional de um dispositivo de hardware em que o ambiente de teste pode ser implementado juntamente com o dispositivo. Esta metodologia proporciona ao engenheiro de verificação toda a base do ambiente de teste [10].

A metodologia é baseada no conceito de cobertura funcional, na qual a verificação é finalizada quando a cobertura funcional for atingida [2]. A cobertura funcional é composta por um ou mais *buckets* (implementados pela classe bve_cover), que dependem de cada aplicação a ser verificada e possuem as funcionalidades que deverão ser estimuladas além da quantidade de vezes que elas deverão ser realizadas. No momento em que a cobertura funcional é atingida, o dispositivo foi verificado e a geração de estímulos pode ser finalizada [2].

A metodologia possui cinco componentes principais que compõem o ambiente de testes (*testbench*) além do DUV, que é o dispositivo a ser verificado. Os módulos que integram a metodologia VeriSC são: Source, TDriver, Modelo de Referência (MR), TMonitor e Checker. A figura 2 demonstra o testbench composto pelos cinco módulos da metodologia e pelo DUV.

O módulo Source é responsável pela geração dos dados que serão utilizados para verificar o DUV. Esse módulo também possui a função de enviar os dados para o MR e TDriver.

Os módulos TDriver e TMonitor possuem a função de modificar os tipo de dados para serem utilizados pelos próximos módulos . O TDriver transforma os dados do nível de transação para o nível de sinais e os encaminha para o DUV. O TMonitor recebe os dados no nível de sinal do DUV e os transforma para o nível de transação, para serem utilizados pelo Checker.

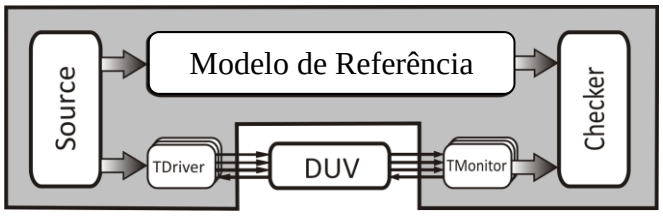

Fig. 2. Fluxo de dados da metodologia VeriSC que apresenta os módulos da metodologia em conjunto com o DUV.

O MR possui as mesmas funcionalidades que o DUV, porém ele pode ser escrito em qualquer linguagem de programação e pode ser implementado simultaneamente com o DUV.

O Checker é o módulo que recebe as respostas geradas pelo DUV e MR e avalia se elas são idênticas ou diferentes. Caso as respostas sejam diferentes, o Checker envia uma mensagem de erro informando o valor esperado e o valor recebido. Se as respostas forem idênticas, o processamento dos dados de entrada gerou os mesmos dados de saída no DUV e MR.

IV. METODOLOGIA VERISC COM ALGORITMOS GENÉTICOS

A metodologia VeriSC utiliza a biblioteca SystemC Verification Library (SCV) para realizar a geração dos dados para a verificação funcional. Contudo, esta biblioteca realiza a geração dos dados de maneira pseudoaleatória. A biblioteca SCV gera, em todas as execuções da verificação, os mesmos dados estimulando as funcionalidades sempre com os mesmos valores.

Essa característica da biblioteca SCV dificulta a busca de erros do dispositivo. A geração de dados se assemelha a uma busca. O objetivo dessa busca é encontrar todos os dados que estimulem as funcionalidades do dispositivo para verificar se

a resposta obtida é uma resposta válida. Gerar sempre os mesmos dados em todas as execuções significa percorrer sempre os mesmos caminhos nessa busca. Já a geração de dados diferentes permite uma busca mais vasta, percorrendo caminhos diferentes em cada execução.

A metodologia VeriSC associada com a geração de dados utilizando algoritmos genéticos possibilita que a verificação funcional seja mais robusta, guiando a busca por caminhos diferentes a cada nova execução.

O uso dos algoritmos genéticos nessa metodologia modifica os módulos Source e Checker. O módulo Source encaminha os dados gerados pelo AG para serem processados pelo MR e DUV. O Checker implementa uma realimentação que fornecerá as respostas dos dispositivos para serem analisadas pelo AG e, assim, o algoritmo aprimora os dados em cada novo ciclo. A figura 3 apresenta a metodologia VeriSC com a adição do AG e da realimentação.

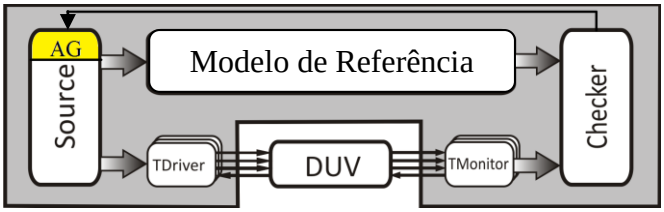

Fig. 3. Fluxo de dados da metodologia VeriSC adicionado a geração de dados feita pelo AG e a realimentação.

V. TEMPLATE

A implementação do AG foi realizada em duas classes: indivíduo.h e algoritmogenetico.h. Ambas as classes foram programadas na linguagem de programação C++ para que seja possível a comunicação entre a essas classes com os módulos da metodologia VeriSC, que foram programados na linguagem SystemC.

A classe indivíduo implementa o cromossomo de cada solução candidata presente na população e, consequentemente, cada posição do vetor corresponde a um gene do cromossomo. A figura 4 apresenta os métodos da classe indivíduo.

A classe indivíduo possui três métodos que definem as operações que podem ser realizadas em cada gene do indivíduo. Esses métodos compreendem a inicialização de valores em cada gene dos indivíduos, a alteração de um valor nos genes e a resposta de uma requisição do valor de um gene. No exemplo da figura 4, cada variável do tipo vetor possui 100 posições, o que caracteriza que um cromossomo de um indivíduo possui 100 genes.

```
class individuo{
 double indiv[100];
 public:
 void inicializa_individuo(int quant_indiv)
 void set_gene(int posicao, double valor)
 double get_gene(int posicao)
};
```

Fig. 4.Trecho de código que apresenta a variável e os métodos implementados pela classe indivíduo.

A classe *algoritmogenetico* possui três vetores que armazenam variáveis do tipo indivíduo, constituindo uma estrutura semelhante a uma matriz. A variável *população* é a variável principal, aquela que armazenará os indivíduos da população atual e os indivíduos descendentes. As variáveis *resultados* e *nova_pop* são utilizadas apenas para auxiliar a manipulação dos indivíduos durante os métodos de avaliação, ordena população e nova_população. A figura 5 mostra as variáveis e os métodos implementados pela classe *algoritmogenético*.

```
class algoritmogenetico{
        individuo populacao[100];
        individuo resultados[100];
        individuo nova_pop[100];

        public:
        void construtor()
        double AG_get_gene(int posicao_indiv, int posicao_gene)
        void set_individuo(int posicao_indiv, int posicao_gene, int valor)
        individuo retorna_individuo(int posicao_indiv)
        void inicializa_populacao()
        void avalia_populacao_bucket1(int posicao_indiv, int variavel_1)
        void avalia_populacao_bucket2(int posicao_indiv, short vetor_1[64])
        void ordena_populacao()
        void nova_populacao()
};
```

Fig. 5.Trecho de código do classe algoritmogenetico que apresenta as vetores de armazenamento dos indivíduos e os métodos implementados.

Os quatro primeiros métodos implementados por esta classe realizam funções básicas de uma estrutura de dados, como seu construtor, retorno de uma posição da matriz ou de uma linha da matriz e a alteração de um valor em uma posição da matriz.

Os métodos seguintes implementam as características do AG. Primeiramente, a população de indivíduos é inicializada com valores aleatórios através do método inicializa_populacao.

Em seguida, são apresentados dois métodos (avalia_populacao_bucket1 e avalia_populacao_bucket2) responsáveis por descrever quais são as funções de aptidão utilizado pelo AG. Na primeira função de aptidão, é avaliado se um gene do indivíduo corresponde ao valor desejado. A função recebe dois parâmetros para a avaliação, o primeiro é qual indivíduo será avaliado e o segundo é o gene que será avaliado pela função. Na segunda função de aptidão, é avaliado um vetor de números (declarados como *short*) de todos os indivíduos, avaliando um por vez.

O próximo método ordena_pop é o responsável por ordenar toda a população em ordem decrescente. A ordenação é realizada analisando o valor de aptidão de cada indivíduo e posicionando os mais bem avaliados nas primeiras posições.

O método nova_população contém o método de seleção de indivíduos pais e os operadores genéticos do AG. Após a ordenação, este método seleciona os indivíduos que irão conter a população de indivíduos pais e aplica os operadores da recombinação e da mutação, gerando a nova população de indivíduos.

Foram implementados dois métodos de seleção, o método da roleta e do torneio. A reprodução é realizada através da recombinação de um ponto com chance de ocorrência de 90%. O processo da mutação possui a probabilidade de ocorrência de 5% e altera o dado aleatoriamente.

A mudança presente no módulo Source é a adição do sinal de realimentação e das chamadas aos métodos da classe algoritmogenetico para a geração dos dados. No módulo Checker é realizada a adição de um sinal que fornece os valores dos resultados dos processamentos recebidos por ele, para serem avaliados pela função de aptidão no Source.

VI. RESULTADOS E DISCUSSÕES

O *template* foi testado em um módulo de um decodificador de áudio e vídeo como estudo de caso, o MPEG4 [16]. Foram analisados em quais parâmetros e métodos haviam a necessidade de modificação para se adequar à solução do problema.

Na classe indivíduo, a única modificação necessária é relacionado ao tamanho do vetor desejado. Cada problema possui a sua quantidade de variáveis distintas e, assim, a quantidade de genes em um cromossomo varia de acordo com o problema.

Na classe algoritmogenetico, o tamanho dos vetores representam a quantidade de indivíduos em uma população. Este parâmetro não possui a obrigatoriedade semelhante da classe anterior, podendo ser alterado de acordo com o desenvolvedor do sistema.

Os métodos construtor, AG_get_gene, set_individuo e retorna_individuo são métodos padrões da estrutura de dados e não necessitam de mudanças.

O método inicializa_população realiza a inicialização de todos os genes com valores aleatórios. Neste método é necessário a declaração da faixa de valores de cada gene, pois um gene pode conter valores booleanos (0 ou 1), valores inteiros, valores reais e etc. A declaração de valores depende dos tipos das variáveis utilizadas para resolver o problema.

A função de aptidão é o método que menos padronizado de todo o AG. Como essa função mapeia o problema para o AG, ela está diretamente relacionada com a descrição, análise e objetivos do problema. O problema pode ser mapeado em apenas uma função ou várias. Esses fatos justificam o motivo de não ter sido realizado um padrão para esta função. Porém, foi padronizado os cabeçalhos do(s) método(s) e exemplificados na figura 5, na qual a primeira função avalia um gene dos indivíduos, enquanto que na segunda é avaliado um vetor contendo 64 números.

O método ordena_populacao ordena a população de forma decrescente e não necessita de nenhuma modificação. Ele apenas recebe a população de indivíduos, analisa seu valor de aptidão e ordena todos os indivíduos.

O método nova_pop aborda o método de seleção e os operadores genéticos do AG. Foram realizadas duas implementações distintas para o método de seleção, o método da roleta e do torneio. É necessário a escolha de uma dessas implementações para ser o método de seleção. O método da roleta gera as parcelas da roleta automaticamente, sem a necessidade de nenhuma modificação. Já o método do torneio, se não houver a necessidade de modificar a quantidade de indivíduos selecionados para o torneio, nenhuma mudança é necessária. O operador de recombinação implementado é a recombinação de um ponto e o operador da mutação modifica o valor de um gene aleatoriamente (dentro da faixa de valores determinada no método inicializa_populacao). Nos operadores genéticos, as modificações podem ser realizadas na probabilidade de ocorrência de cada operador.

O *template* desenvolvido possui a característica de otimizar a geração de dados para dispositivos de *hardware* que a partir de uma entrada, gera uma saída. A evolução do AG consiste em receber e analisar uma resposta para cada estímulo fornecido.

VII. CONCLUSÕES

Utilizando o conceito de algoritmos genéticos, foi criada uma nova forma de geração de dados que foi inserida dentro da metodologia VeriSC por meio de um *template*.

Esse *template* possui vários métodos que não necessitam de modificações para o seu funcionamento. Os métodos que necessitam de mudanças são aqueles que dependem necessariamente da descrição do problema a ser resolvido.

A utilização de um *template* reduz o tempo gasto durante um projeto de *hardware*, facilitando a verificação do *hardware* para o engenheiro de verificação e, ao mesmo tempo, realizando uma verificação mais robusta.

AGRADECIMENTOS

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), e à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás (EMC/UFG).

REFERÊNCIAS

[1] J. Bergeron, *Writing tesbenches: functional verification of HDL models*, Springer, 2ª Edição, Nova Iorque, 2003.

[2] K. R. G. da Silva, "Uma metodologia de verificação funcional para circuitos digitais", Doutorado tese, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, Brasil, 2007. Acedido em 08 de Junho de 2014, em: http://lad.dsc.ufcg.edu.br/uploads/Lad/tese_karina.pdf

[3] A. E. Eiben e J. E. Smith, *Introduction to Evolutionary Computing*, Springer, 1ª Edição, Nova Iorque, 2003.

[4] R. Linden, *Algoritmos Genéticos*, Ciência Moderna, 3ª Edição, Rio de Janeiro, 2012.

[5] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, 28ª Edição, Crawfordsville, 2006.

[6] J. C. H. de Barcellos, "Algoritmos genéticos adaptativos: um estudo comparativo" Mestrado dissertação, Departamento Escola Politécnica, Universidade de São Paulo, Brasil, 2000.

[7] V. Subedha and S. Sridhar, "An Efficient Coverage Driven Functional Verification System based on Genetic Algorithm," *European Journal of Scientific Research*, vol. 81, no. 4, pp. 533–542, 2012.

[8] J. H. Andrews, T. Menzies, F. C. H. Li, "Genetic algorithms for randomized unit testing", *IEEE Transactions On Software Engineering*, vol. 37, no. 1, pp. 80-94, Janeiro/Fevereiro 2011.

[9] A. Samarah, "Automated Coverage Directed Test Generation Using a Cell-Based Genetic Algorithm", The

Departament of Electrical and Computer Engineering, Concordia University, Montreal, Canada, 2006.

[10] K. R. G. da Silva, E. U. K. Melcher, G. Araujo, "An Automatic Testbench Generation Tool for a SystemC Functional Verification Methodology," *17th Symposium on Integrated Circuits and Systems Design*, pp. 66-70, Setembro 2004.

[11] G. Di Guglielmo, L. Di Guglielmo, F. Fummi, G. Pravadelli, "Efficient generation of Stimuli for functional verification by backjumping across extended FSMs," *Journal of Electronic Testing*, vol. 27, no. 2, pp. 130-162, Março 2011.

[12] A. Pozo, A. F. Cavalheiro, C. Ishida, E. Spinosa e E. M. Rodrigues, "Computação Evolutiva". Acedido em 08 de Junho de 2014, em: http://www.inf.ufpr.br/aurora/tutoriais/Ceapostila.pdf.

[13] H. F. de A. Oliveira, "BVM: reformulação da metodologia de verificação funcional VeriSC", Departamento Engenharia Elétrica, Universidade Federal de Campina Grande, Brasil, 2010.

[14] A. Ferreira, R. Franco, and K. da Silva, "Using genetic algorithm in functional verification to reach high level functional coverage,". Acedido em 08 de Junho de 2014, em: http://www.inf.ufrgs.br/sim-emicro/papers/sim2013_submission_50.pdf.

[15] Q. Guo, T. Chen, H. Shen, Y. Chen, W. Hu, "On-the-fly reduction of stimuli for functional verification," *2010 19th IEEE Asian Test Symposium*, pp. 448-454, Dezembro 2010.

[16] G. Silveira, K. da Silva, and E. Melcher, "Functional verification of an MPEG-4 decoder design using a random constrained movie generator," *SBCCI '07 Proceedings of the 20th annual conference on Integrated circuits and systems design*, pp. 360-364, Setembro 2007.