We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 5

Functional Verification of Digital Systems Using Meta-Heuristic Algorithms

Alfonso Martínez-Cruz, Ignacio Algredo-Badillo,
Alejandro Medina-Santiago,
Kelsey Ramírez-Gutiérrez,
Prometeo Cortés-Antonio,
Ricardo Barrón-Fernández,
René Cumplido-Parra and Kwang-Ting Cheng

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.80048

Abstract

Trends in technological developments, such as autonomous vehicles, home automation, connected cars, IoT, etc., are based on integrated systems or application-specific integrated circuits with high capacities, where these systems require even more complex devices. Thus, new techniques to design more secure systems in a short time in the market are needed. At this point, verification is one of the highest costs in the manufacturing stage and most expensive in the design process. To reduce the time and cost of the verification process, artificial intelligence techniques based on the optimization of the coverage of behavioral areas have been proposed. In this chapter, we will describe the main techniques used in the functional verification of digital systems of medium complexity, focusing especially on meta-heuristic algorithms such as particle swarm optimization, genetic algorithms, and so on. Several results are presented and compared, where the opportunity areas will be described.

Keywords: digital systems, meta-heuristics, FPGAs, PSO, genetic algorithms, functional coverage, verification, automation

1. Proposed techniques for functional verification of digital systems

New applications in different areas, such as the automotive industry, robotics, IoT, and smartphones, among others, require increasingly complex digital devices. This implies the use of new techniques to reduce the design time of the devices, ensuring useful functionality according to the specification. It is important to know that manual simulation and functional verification require much time and expertise, so it has been necessary to develop software tools that improve performance, reduce manufacturing times, decrease verification costs, and increase the confidence level of the RTL implementations. In addition, new systems use a large amount of computational resources and new algorithms that increase the complexity of digital systems and require new methods to analyze and evaluate the device under verification (DUV). Several works of researchers on functional coverage methods have been made. Most studies use the following philosophies: static (methods based on logical or mathematical techniques), dynamic (methods based on simulation), and hybrid methods (combining static and dynamic). Next, works based on meta-heuristic and data mining algorithms report different methods for verification.

To perform verification of digital systems, different approaches have applied heuristic algorithms, for example, genetic algorithms (GA) that apply the evolution theory, where individuals within a population adapt to the conditions to the environment, compete for resources, and generate the evolution of the population through operators such as selection, crossing and mutation. Most of the time, the generation of pseudorandom tests produces worse results than this generation of test sequences. For example, authors in [1] perform a PowerPC architecture verification using genetic algorithms by generating pseudorandom custom instructions and encoding a sequence of instructions with a fixed length. The population size is small to reduce system simulation time. In the same way, in [2] the authors presented an implemented method to generate directed tests through a genetic algorithm, and a cell represents the chromosome in a uniform random distribution in two limits; the different parameters of the method were not fully automated; therefore, extensive knowledge of the evolutionary framework by the user is needed. In addition, in [3], the authors configure a genetic algorithm, which is included in a software platform to improve the functional coverage in a device. In this latter, chromosome coding is based on established instructions, and the proposed method helps to achieve uncovered tasks and increases the hit rate in the test of hard cases, improving the results of the pseudorandom test generation.

Some works have implemented ant colony optimization (ACO) and particle swarm optimization (PSO). On the one hand, the ACO uses the imitation of the behavior of ants seeking better paths from the initial place to the food place; ants get their food by means of pheromones and, in this way, other ants walk the paths and can provide positive feedback. In [4], a method based on the ACO that combines the pseudorandom test generation with a software platform that generates the digital system states is presented. The results show a reduction in computational complexity compared to random generation and other heuristics based on GA. On the other hand, the PSO algorithm is based on the interaction between the particles in the swarm. For instance, the authors in [5] present a verification method by using branches as a coverage metric and a PSO algorithm to perform the validation of RTL implementations.

Other algorithms have been applied; for example, in [6], the authors used Bayesian networks in a functional verification method, and this type of networks is a model based on probabilistic graphs that are composed of random variables or nodes and edges that represent dependencies between them. The verification has feedback and the ability to cover hard cases and increase the coverage rate of progress, even though a manual configuration for the process was required. Other techniques were proposed for hardware verification based on meta-heuristics [7], where a differential evolution (DE) algorithm is applied; the verification is based on a coverage model using *coverpoints* and the algorithm is used to generate test vector sequences.

Works have improved the functional coverage using data mining. In [8], the authors proposed a learning methodology where knowledge from test is extracted. The extracted data is reused to generate tests with similar values to other important ones and cover new assertions. The method is applied to perform a constrained random verification of a processor and reports improvements in assertions coverage through the information extracted in the verification. The authors in [9] proposed an automatic learning method of rules regarding micro-architectural behavior of the instructions, and these rules were embedded in a stimuli generator tool. The method is applied in a microprocessor, improves the quality of the test cases generated and reaches interesting coverage events. In addition, [10] describes a method based on decision trees. In this method, before activating the sentences, they go through an engineering of formal verification to filter the candidate alterations in the output, generating automating RTL sentences. The proposed method was divided into two spaces: static and dynamic techniques. Static analysis techniques were used to direct the data mining process. In addition, Hidden Markov Models (HMM) are statistical methods that use probability measurements for sequential models of the data represented by sequences.

Other techniques used in functional verification are based on mutations that are changes of the RTL implementation, and such coverage metrics are used to drive the verification progress during simulation. For example, in [11] the authors proposed a methodology to verify a microprocessor using mutations. To test the vector sequences, the design simulation is performed first, then a set of mutations is added, and the verification is executed. Finally, a comparison of the results is made. One of the problems occurs when a large number of mutations are added because the verification time is increased.

In this chapter, an alternative hybrid method that uses coverage models is presented. This method represents the device behavior through CoverPoints, and fitness functions focused on sets of specific behavioral regions. In particular, a PSO algorithm with a re-initialization mechanism (BPSOr) is described. The method represents a hybrid technique that uses a simulation tool and meta-heuristic algorithms through a proposed verification interface.

2. Functional verification elements

In large-scale electronic integration design, functional verification is the verification process of a design logic that complies with specific rules design for its operation and manufacturing in an integrated circuit. The functional verification answers the question: “Does the proposed

electronics design meet the desired design and functionality requirements?" A complex task with times and high computational efforts is presented mainly in VLSI design. The functional verification is adjacent to a deeper design verification that, in addition to functional verification, adopts nonfunctional aspects such as time, design and power, implemented in the design of mixed circuits for signal processing.

There are different elements which work during the functional verification process. A verification system usually consists of several types of components:

1. Test generators are used in the stages of the functional verification where the test vectors are used to detect a fault presented in the specifications and the generation of the code. These generators use a full SAT type of NP resolution that is computationally expensive. In other types of generators, the vectors are created manually, for instance, the patented graphics-based generator (GBM). In short, modern generators create random vectors that are applied statistically on the design verification. Therefore, the users of the generators do not clearly specify the requirements to the test generation.
2. The supervisors interpret the stimuli produced by the vector generator for the DUV inputs. Generators create entries with a high level of abstraction, for example, transactions or instructions in assembly language. Supervisors convert this entry into inputs for the DUV as defined in the design interface specification.
3. The simulators (software tool) excite the circuits under verification to obtain their outputs, depending on the current state of the design and the input vectors injected (verification vectors). In this case, the software tool has a description of the design network list.
4. The monitor converts the state of design and its outputs into an abstraction transaction level that will be stored in a score-board database for later verification.
5. The verifier validates the score-board data. In some cases, the generator produces the expected results, in addition to the inputs. For those cases, the verifier must validate actual results that match the expected results.
6. The supervisor is included in the verification environment and manages all the above components together.

Figure 1 shows a pseudorandom test generation scheme where the functional coverage is used as coverage metric. The verification is done using constraints for the stimuli during the device simulation. After a specific number of iterations, the coverage information is reviewed by

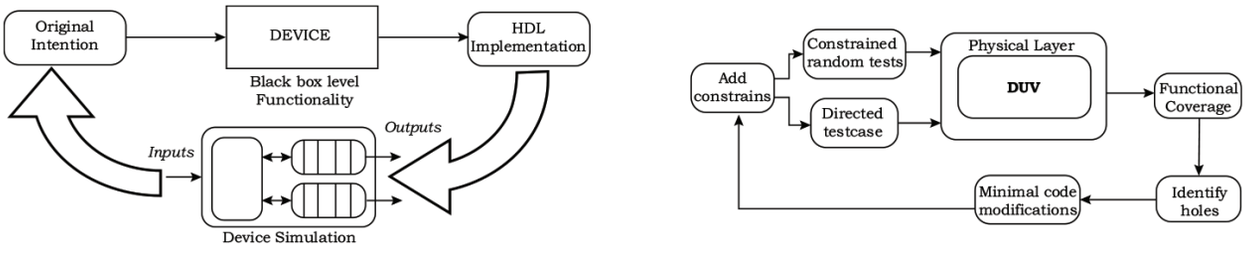

Figure 1. HDL verification through pseudorandom test generation.

http://dx.doi.org/10.5772/intechopen.80048

identifying the holes produced and, then adding more constraints. Finally, the process is executed until a stop criterion is met.

Verification is a very difficult task due to a large volume of possible test cases that exist even in a simple design. The verification can be attacked by many methods:

1. The logical simulation executes the logic of a circuit before building it to obtain its approximate behavior.
2. Simulation acceleration applies special-purpose hardware to the logic simulation problem.
3. Programmable logic creates a version of a system; this is expensive and even much slower than real hardware and orders of magnitude faster than simulation. For example, they can be used to start the operating system in a processor.
4. Formal verification attempts to prove mathematically that certain requirements are met or that certain undesired behaviors cannot occur.
5. Automated verification uses automation to adapt the test bench to changes in the register transfer level code.
6. Specific HDL versions and other heuristics are used to find common problems.

Different methodologies have been proposed in order to perform the functional verification. Three different philosophies have been suggested in order to perform the functional verification: static methods (formal methods), dynamic methods (which are based on simulation) and hybrid methods (which does not fall in formal and informal methods). Every philosophy contains different strategies in order to test the digital system functionality. For example, formal methods perform the verification using mathematical expressions to give a formal description of the device's behavior. Examples are model checking, theorem proving, etc.

During the verification based on dynamic methods, the stimuli are used to exercise the functionality, and test benches are also implemented and added to the verification environment. These methods are very scalable and practical. Due to the greater constant complexity of the devices, the use of these methods in the industry is very common. On the other hand, even if the designs are completely verified, it is not easy to guarantee that there are no errors.

Hybrid methods make up the third category, combining the formal and dynamic techniques. This type of methods is focused on increasing the coverage obtained from the bottleneck guiding the search through the full coverage space. A disadvantage is that its design requires broad background about verification techniques.

2.1. Problems solution through meta-heuristic algorithms

Searching directly for test vectors sets that appropriately evaluate and examine the functionality of the developed devices is not trivial. For example, for the deterministic methods, the consumption of resources is generally growing exponentially, which depends on the size and architecture of the circuit. Consequently, other solutions have been proposed, i.e., methods that use meta-heuristic are mainly applied to decrease the computational complexity when verifying the device.

Meta-heuristics methods are algorithms to find a global solution using local approximations and heuristics. A meta-heuristic represents a top-level strategy which guides the heuristics to solve a problem. Frequently, not all search details are specified and can be adjusted according to a specific problem. Alternatively, there are general techniques to handle the directed search where optimal local solutions will be avoided; they are employed in the verification context. For instance, in genetic algorithms, a population of individuals is used as an initial set of solutions. The fitness value of a test sequence represents how good an individual is. In addition, the search for solutions is directed by an individual's combination that uses a set of operators.

There are different definitions of meta-heuristics; commonly, a meta-heuristic can be defined as a process that drives other heuristics through a combination of elements to explore and exploit the search spaces. Besides that, it uses learning strategies to manage the information obtained and achieve optimal solutions. Some examples of meta-heuristics are ant colony, artificial bee colony algorithm, genetic algorithms, etc. Many works have used this type of algorithms to find solutions to different problems. Its applicability is suitable in optimization problems where the computation of cost functions is so expensive and influenced by a type of noise. Consequently, meta-heuristics are techniques that find good solutions in large search spaces.

2.2. Automated functional verification in digital systems

In this work, the functional verification of the devices is designed and executed automatically. Moreover, when the functional verification uses the coverage data that is produced from each simulation, it is named as "directed functional verification." A fundamental aspect is the coverage information (integrity measurement) for the test sets and represents the data where the revision is made in the verification. In addition, the analysis of this process allows the generation of new test sequences to evaluate other coverage regions.

The verification by simulation of the device is carried out when the expected functionality is translated into the implementation of RTL according to the specification and the criteria of the designer. Then, the device is reviewed through a series of steps, for example, checkers, monitors, test-benches, etc. In the end, the verification platform gives the coverage results that express the percentage of functionality verified. When reviewing the functional verification definition in [12], the RTL implementation of a device based on a set of features and operational requirements should be provided, to execute the verification, which is composed of the process that guarantees the device implementation that complies with each feature given in the specification.

Automated functional verification involves different elements such as coverage models, control flow graphs, test sequences, and cost functions, among other elements. When these elements interact, a system of test vector generation is formed. Some verification methods use this type of scheme to perform verification of digital devices.

An important case occurs when the test generation uses feedback information to explore new behavior regions, when this happens it is named as coverage-directed test generation. There are different definitions, for instance, according to [12], this generation allows to produce different test sequences to exercise different functionalities (characteristics of the coverage

models) of the device. Therefore, this process occurs when a test sequence is injected into the input of a device and then a new function value is exercised. Then, the value obtained is stored. Finally, all device states are reviewed and a new test is generated.

In other words, first, a test vector sequence is injected into the input of the device; then, if a new feature from the intended behavior is covered, the test sequence and the value of the feature exercised are stored. Later, the device states are reviewed and another test vector is produced to verify the "DUV." After this, all the states are verified and the values of coverage metrics are analyzed.

Figure 2 shows the main steps in the automation of the directed test generation. In this scheme, a verification plan based on based on a functional specification is needed, which describes what characteristics of the device will be verified and how it will be done.

2.3. Functional coverage models and coverage metrics

A functional coverage model can be described as a functional coverage space where the device behavior is captured. This means that it represents a coverage space that contains the interrelationships that exist between inputs, outputs, tasks, events, conditions and characteristics, which could show the correct functionality of a device with a confidence degree of a device. The coverage model is designed based on the implementation or device specification and a coverage metric or coverage structure.

A coverage metric consists of a heuristic to measure what part of the device behavior has been verified correctly. The main objective of this measure is to reflect which parts of the functionality have been met with correct execution during the processing of the information by the device, i.e., functional coverage (verify that all characteristics meet the specification), statement coverage (verify if the lines of code in the HDL implementation are exercised), branch coverage (analyze if the paths are traveled through the branches during the simulation), and finite-state machine (check how many states have been covered correctly).

The models are fundamental components of the verification process. A coverage model using stimuli, events, constraints, and CoverPoints is generated. It means that the coverage models are representations that map the intended behavior through characteristics, inputs, outputs, and its interrelations. A coverage model can be based on coverage points (CoverPoints). CoverPoints represent the values of each variable in a coverage model.

A coverage model can be defined as the different characteristics to represent the device behavior according to a functional specification that has different constraints. In particular,

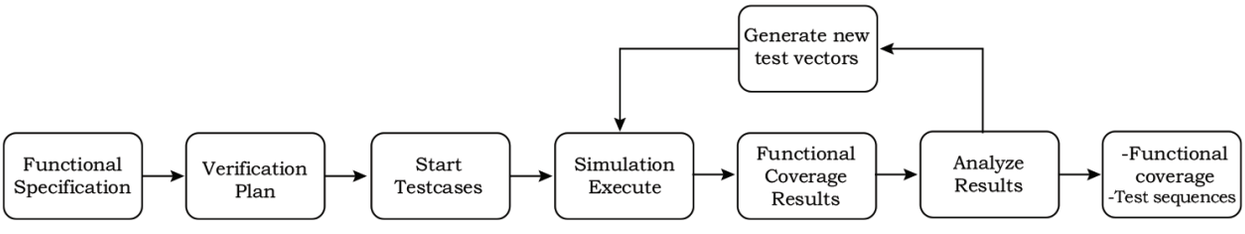

Figure 2. Automation of directed test generation.

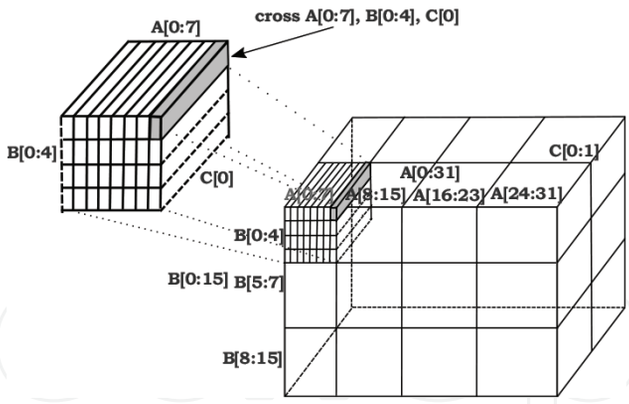

Figure 3. Functional coverage model.

the way of representing the behavior affects the granularity of the model, that is, a model with more characteristics can represent the original intention more efficiently, and as a consequence, it has a higher level of granularity. The accuracy of that model describes the implementation. The coverage model may contain explicit and implicit device behavior features. Moreover, the models are designed according to the device specification and implementation. **Figure 3** shows a coverage model where fidelity of a model determines how closely the model defines the actual requirements of a device behavior.

3. Verification method using BPSOr algorithm

The proposed verification method uses the BPSOr algorithm, which is based on several psychological aspects and social elements. In this social-cognitive context, individuals must interact among them, where the best performance occurs within the particle group and previous behaviors. Each individual is a particle, each particle group is a neighborhood, and each cognitive and social particle behavior is influenced by an improved performance from the groups.

At this point, two proposals are presented: *lbest* (local-best) and *gbest* (global-best). In the first proposal, the particle with the best performance in its groups affects to the remaining parts. In the second proposal, the swarm is important, because particles are connected among them, where the best performance of a particle from the swarm affects it and the results are improved.

In the swarm, each dimension is analyzed, and there are two main computational problems: memory and velocities; the first one establishes the best particle location, comparing the actual position and other better ones by means of the search. In addition, a key metric is the rate of change, which is computed for the particle based on the velocities to obtain *gbest* (the best global) and *lbest* (the best local solution). Incremental changes in both learning and attitude are simulated, providing the granularity of the search in the problem space. On the one hand, speed represents changes in probabilities, which may have the value "1" or "0." On the other

hand and considering the particle dimension, the attitude of the changes represents the probability, which can be "1" or "0." For these reasons, the sigmoid function $S(V_{id})$ [13] transforms velocities to probability values and obtains a zero state for each particle, see Eq. 1. If v_{id} is high, the particle bit will probably be 1, and if the v_{id} is low, the particle bit will probably be 0, where v_{id} is a value in the range $[V_{min}, V_{max}] = [0.0, 1.0]$, ensuring that two possible values take the dimension bit (for the sequences): "1" or "0."

$$S(V_{id}) = \frac{1}{1 + exp\,(-v_{id})} \tag{1}$$

It is important to control the influence (from paths by each particle and other particles in the population), because the particles can move to the regions where the fitness variables have the best values. In this case, the pseudorandom values have produced better results when they are the mutation operators in the genetic algorithms. If the PSO algorithm is expressed in real numbers, a great number of problems are presented in binary domains, requiring extra operations for converting real values to binary values.

In binary versions, the PSO algorithm uses binary data directly with a re-initialization process, see **Algorithm 1**. The latter is composed of instructions or rules, where a particle is represented by a set and its elements are binary sequences. In this algorithm, in the first step, the position $\vec{x_i}$ and velocity $G\left(\vec{xi}\right)$ are initialized and computed for each particle. In the second step, $G\left(\vec{xi}\right)$ and its best previous position p_{id} are compared. If $G\left(\vec{xi}\right)$ is better, then its best position p_{id} is equal to x_{id}. In this case, the velocities v_{id} are compared. For every particle dimension, x_{id} has a value "0" when ρ_{id} position fitness is less than $s(v_{id}(t))$ (from sigmoidal speed function), but it has a value "1." These steps are executed until stop condition is reached.

Algorithm 1. Pseudocode of Binary PSO with a re-initialization process (BPSOr).

```
Swarm Initialization.
while No termination condition is reached do
    while No termination condition for coverage percentage is reached do
        for i=1 → Particle_Number do
            Compute fitness values: G(x⃗_i) and G(p⃗_i)
            If G(p⃗_i) is less or equal than G(x⃗_i), then the new best position p_id is x_id.
            for Each neighbour do
                Compare the best performance of population G(p⃗_g) with the performance of
                  each particle in the neighborhood G(p⃗_j). If there is a better performance than
                  the best global then replace it.
                Compute the current velocity v_i(t) and constraint it according to V_max and
                  V_min.
                For every particle dimension x_id and the Sigmoid function, the assigned
                  output is '1' or '0'.
            end
        end
    end
end
If the global best performance G(globalBest) is greater or equal than the best current performance
  G(currentBest), then the velocities v_id(t) for each dimension particle are re-initialized according
  to the probability P_re.
Best particle is registered.
Swarm is re-initialized.
```

In this pseudocode, V_{max} and V_{min} are constraints of each probability of change, where each position of the particle is considered, and the re-initialization process avoids local solutions and covers new behavior regions. This process is based on population-based measures, and if the best global performance is greater than the best current performance of the swarm, then the swarm of particles is initialized again. Consequently, the best particle position and the best particle of the population are stored. In addition, both the current positions and particle velocities are re-initialized. To re-initialize the velocities, a probability value is computed, whose aim is to avoid a convergence in an optimal local solution.

The main aspect of this algorithm is the decision when the bit string has a value of 1 or 0, which is based on the probability and is defined as a function of personal and social factors, see Eq. 2, where: (a) $v_{id}(t-1)$ is a measure of the current probability (individual predisposition) for the decision of 1 or 0; (b) φ_1 and φ_2 are positive random numbers, which are obtained form an uniform distribution, and they represent predefined upper limits; (c) $r1$ and $r2$ are positive random numbers, which can take some value from 0 to 1; (d) $x_{id}(t)$ describes the current state, when a bit-string d is analyzed for the individual i; (e) t represents the current discrete time, and $t-1$ represents the previous discrete time; (f) p_{id} is the variable that represents the best state and has a value of 1 if the individuals with the best success are located when x_{id} is 1 and 0 in otherwise; (g) p_{gd} is the best neighbor and has a value of 1 if the best success is reached by some number at the moment of examining the neighborhood with state 1 and has a value of 0 in the other case; and (h) ρ_{id} describes a vector or data structure of random numbers, which are obtained by using an uniform distribution among 0.0 and 1.0, and P_{re} represents the re-initialization factor with real value in the unit interval 0.0 to 1.0.

$$v_t(t) = v_{id}(t-1) + r1 \times \varphi_1\left(p_{id} - x_{id}(t-1)\right) + r2 \times \varphi_2\left(p_{gd} - x_{id}(t-1)\right) \tag{2}$$

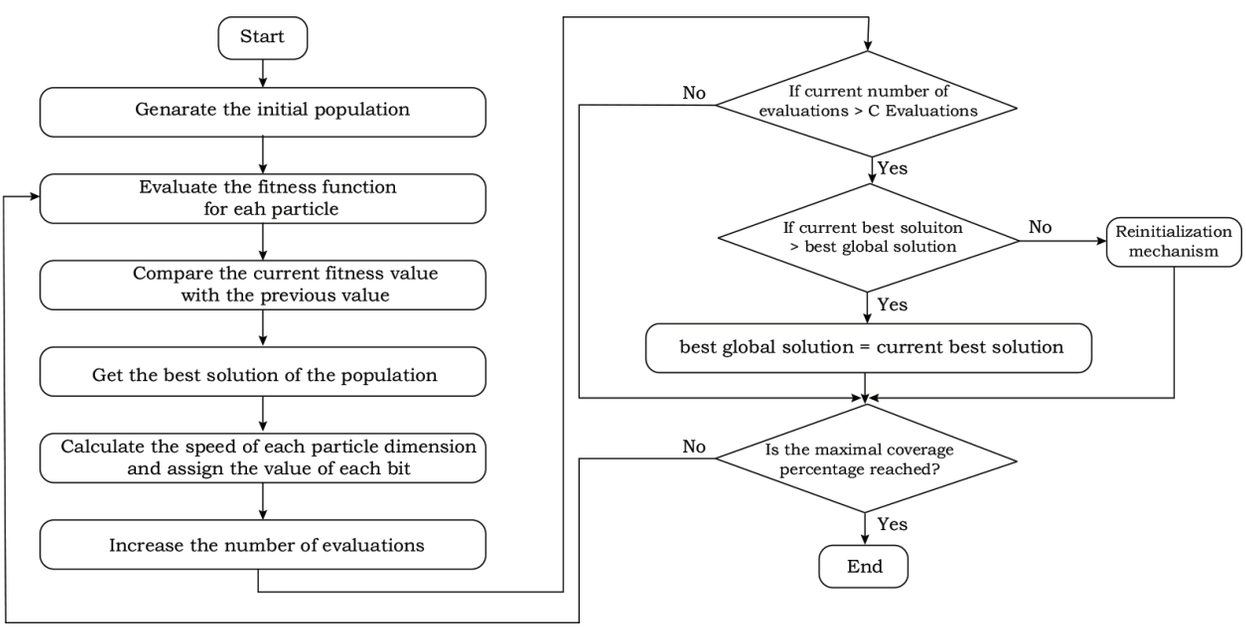

Figure 4. Flow diagram of BPSOr algorithm.

Figure 4 shows the flow diagram of BPSOr algorithm. The different advantages of the binary PSO algorithm with re-initialization (BPSOr) enable to produce test sequences, operating in the verification context and analyzing the devices, which are being verified.

4. Test vector generation method

A proposed interface based on heuristic algorithms and a software tool is used. Moreover, some steps to verify the digital systems are performed. The description of the test generation method implemented in this work is shown in **Algorithm 2**. Firstly, the device parameters must be configured and initiated. In the same way, for the meta-heuristic process, several parameters are initiated and assigned based on the operational requirements (specifications) and implementation. Then, the set of device parameters are initialized.

Algorithm 2. General method of generation of test vector sequences.

```
Initialization of the device under verification.
Initialization of the parameters of the verification method.
Configuration and initialization of the functional verification modules.
Test generation algorithm initialization.
while Stop condition is not met do
    while The coverage criteria reached are not met do
        Generate new test sequences based on the coverage values achieved.
        Evaluate the digital device using a software tool.
        Analyze the information of the values obtained from functional coverage and save the
          statistics.
        Evaluate the fitness function based on the set of specified coverage points.
        Save the best current solution.
    end
end
```

In this case, BPSOr algorithm generates the test sequences; then, a simulation tool to evaluate them is used. The coverage information from device simulation is reviewed and saved. Then, the fitness variables are computed and the best values are stored, which are used in the new iteration.

Local-best topology was implemented in the verification method to perform different experiments. The scheme of this topology is shown in **Figure 5** where test vector sequences are clustering in some sets representing groups of particles; in this case, the particles or test sequences are affected by its fitness value and the best in its neighborhood. The best particle consists of the test sequence with the best fitness value in the group. Additionally, each test sequence or particle can communicate with others in its group. Later, in every iterations, the set of particles is directed toward the best particle in the swarm.

Global-best configuration is represented in **Figure 6**; in this topology, each particle is affected by the best solution in the swarm. All particles are included in the same group and they move toward the best solution. After every algorithm iteration, the test sequence with the best performance guides to the others through the search space.

On the other hand, the fitness function used in the algorithms is shown in Eq. 3. This function is focused on the percentage of holes produced in specific CoverPoints (P_{eh}). Therefore, the

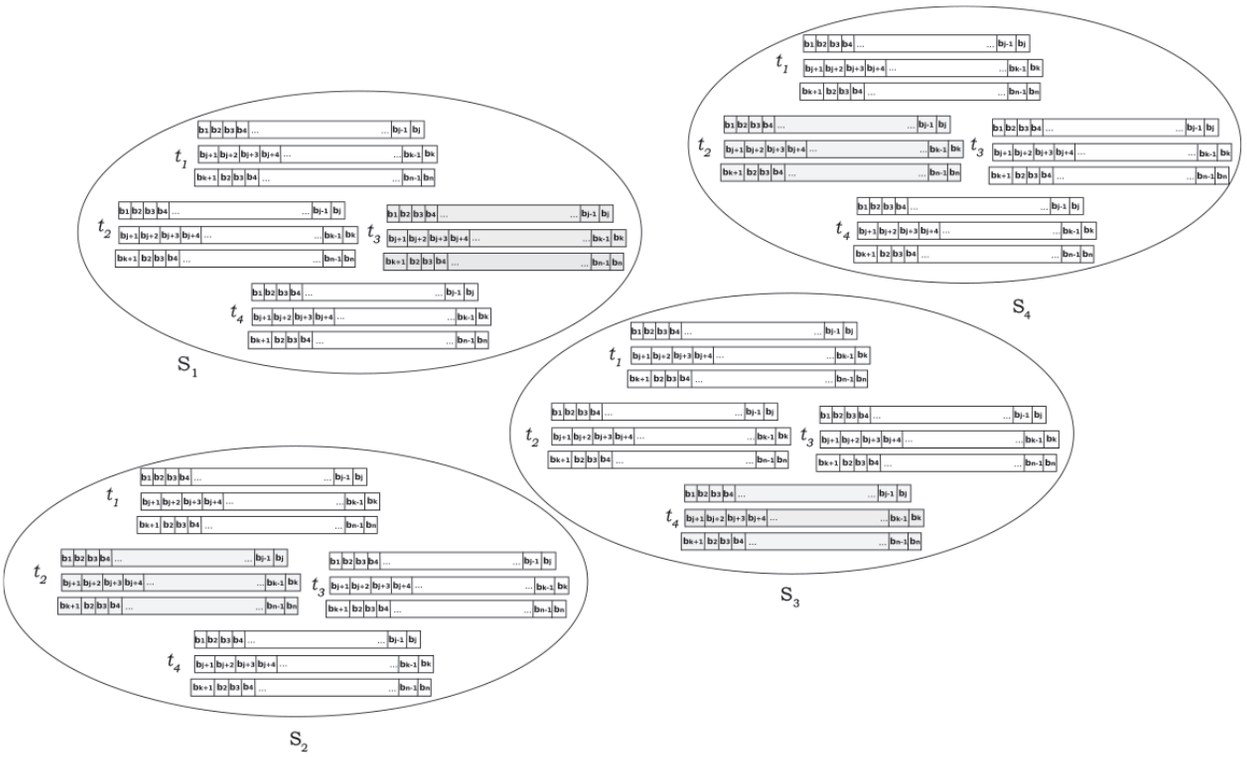

Figure 5. Groups of test vector sequences using the local-best topology.

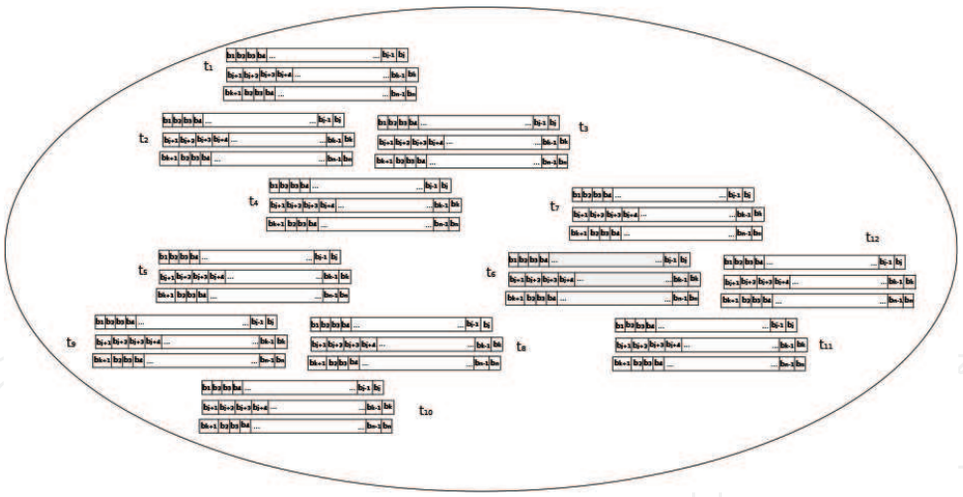

Figure 6. Groups of test vector sequences using the global-best topology.

problem is translated to maximize the number of points covered and, at the same time, minimize the percentage of holes in specific behavior regions.

$$f_1 = MAX\left(\frac{1}{P_{eh}}\right) \tag{3}$$

Test generation sequences are produced in the verification environment to verify the devices. In the beginning, a new binary sequence is tested and analyzed in the device, running the

http://dx.doi.org/10.5772/intechopen.80048

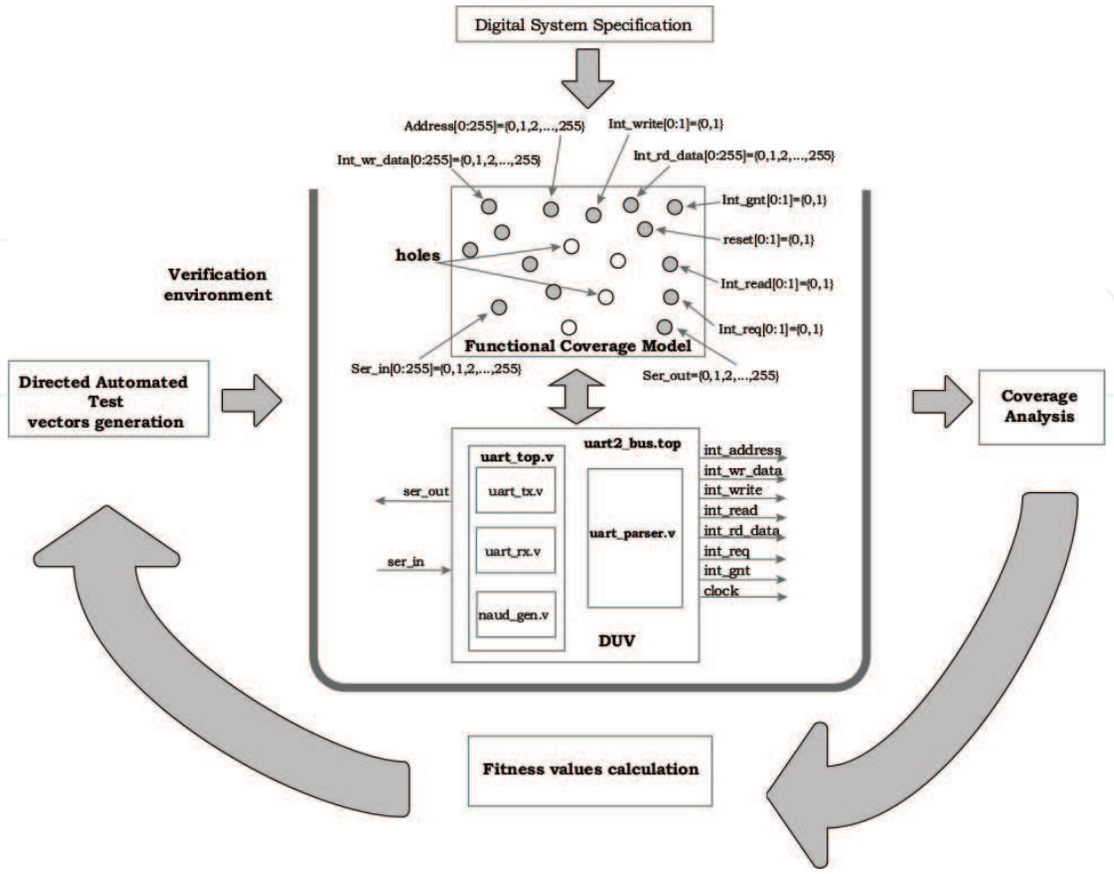

Figure 7. Proposed scheme.

respective simulation. Then, after the last sequence is completed, the cost values are quantified. Their calculation dependents of the points and holes are determined during the respective simulation. The information obtained is delivered to the generator module of test sequences. Therefore, a new sequence is generated and the process is repeated while the stop condition is not reached.

The proposed verification system is composed of several modules that are connected through an interface between C and SystemVerilog languages. **Figure 7** shows a scheme where the system couples the device under verification and the verification process is performed at the RTL level.

5. Case study

The proposed verification method is validated through different experiments using two digital systems. Additionally, the performance of the BPSOr, genetic algorithm, PSO, and random test generation is compared. RTL implementation of the devices was employed as benchmarks in the verification platform. The applicability of this type of method focuses on the block-level verification of IP cores because the automatic verification depends on the controllability degree of events generated from the stimulus during the device simulation. PSO algorithm with a re-initialization mechanism can be more complex computable, however, because this algorithm

achieves fine solutions very quickly, the verification time could be reduced. The best scenarios with different features of BPSOr algorithm will be presented.

Devices such as a UART-IP core were employed in order to perform its verification. The UART-IP can be used as a transmitter and receiver. A 16-bit address bus and an 8-bit data bus are included in the IP core. Its verification was based on the functional specification and the RTL code implementation. The coverage model was implemented using 785 bins in 12 CoverPoints. The initialization and configuration were performed based on the specification. Besides, the verification of a FIFO memory was performed using a coverage model with 784 bins. The memory is often contained in devices such as processors, UARTs, interfaces, and so on. Its implementation was designed in Verilog language and the configuration of the signals was controlled according to the features described in the functional specification.

To develop the proposed experiments, different values of parameters were used, which were included in several scenarios. Therefore, a scenario consists of a set of parameters that are used for the meta-heuristic algorithm. In the case of BPSOr algorithm, the parameters such as topology (global or local), velocity values, number of particles, and ϕ value were modified. Additionally, running a scenario of a defined number of times with a specific parameter configuration is defined as an experiment. The size of the swarm used was among 3 and 16 particles. Also, "global-best" and "local-best" topologies were implemented in the algorithm. The ϕ variable was modified with values from 2.0 to 4.0 for the scenarios. On the other hand, the evaluation of the test sequences was performed using two fitness functions, which are based on the coverage obtained. Basically, these functions get the CoverPoints and the holes generated at the run-time. When the simulation of a device ends, the coverage produced is sent to the test generator module and, finally, a new test is generated.

The results obtained are expressed in the best scenarios where information such as the best, the average, the total iterations, etc. are included. In addition, the binary test sequences were evaluated by modifying their number of elements or length. For example, if a particle is composed of two sequences, then its height is equal to 2.

The obtained results from the best scenarios will be presented to analyze the BPSOr performance. Furthermore, a genetic algorithm (GA) with elitism feature was implemented. Some algorithm parameters such as crossover percentage, mutation percentage, maximal number of evaluations, and population size were modified. A stop criterion was defined using the total number of evaluations. Besides, all CoverPoints were clustered focusing in the points that required to be exercised.

The experiments were performed using a computer with Linux Fedora Core 23. The features of the computer are as follows: Processor model: Intel Core i7-4790 K CPU-4 GHz., RAM: 8 GB, CPU: 4298.5 MHz, and Cache: 8192 KB. Additionally, all experiments were performed over a Linux Fedora Operating System, where the verification platform was successfully installed. After this, the obtained data were saved and reviewed. The obtained results from simulations were handled as statistical information to obtain the best fitness values.

When the verification process is performed, some characteristics could not be exercised due to different factors; for instance, if the behavior regions have the same cost values, then the fitness

functions could not give a difference regarding to other regions. Even, if more algorithm iterations are used, then the test sequences generated will cover the same behavior regions and the holes will not be covered. It is important to design strategies by focusing on the regions that are not easily covered. One strategy consists of a group the CoverPoints in sets with different weights to produce higher behavior areas. In addition, efficient search algorithms are required. Therefore, meta-heuristic algorithms can guide the search usefully and exercise all functionality of the device.

5.1. Experiments

The functional verification method based on meta-heuristic algorithms can test the functionality regions by focusing on specific behavior parts that can required more exploration. In these experiments, the verification method is used to verify two different digital systems. First, to show the performance of the GA a set of experiments will be developed. The genetic algorithm used was a binary version where the best individual remained in the next epoch (elitism). **Table 1** contains the parameters used for three different scenarios. Each experiment was run 30 times and then the coverage percentages and average time were stored. For instance, in the first case, a population of 100 individuals was configured with a crossover of 0.5 percentage and a mutation of 0.001 percentage.

Table 2 shows the obtained results for four best scenarios. Reviewing the results, in the third scenario, a few number of iterations was required in order to reach 100 coverage percentage. Besides, the average time used was 160.46 minutes.

Parameters	**GA scenarios**			
	1	**2**	**3**	**4**
Crossover percentage	0.5	0.5	0.45	0.45
Mutation percentage	0.001	0.001	0.0005	0.003
Population size	150	120	100	100
f	f_1	f_1	f_1	f_1

Table 1. GA algorithm settings for four different scenarios.

Final values	**1**	**2**	**3**	**4**
Best value	100	100	100	100
Worst value	95.44	97.13	99.08	98.30
Average value	98.23	99.12	99.87	99.62
Average evaluations	8090	7944	7353	7623
Average Time (min)	178.09	173.05	160.46	165.36

Table 2. Results obtained using a genetic algorithm in the platform to verify a UART-IP core.

Table 3 shows the four best scenarios using the BPSOr algorithm to perform the verification of a IP-UART core. One of the parameters that was changed is the swarm size. In this case, 3, 6, 9, and 12 particles were used in the proposed method. For example, in the first scenario, the parameters used were: 9 particles, 3 neighborhoods, ϕ = 4.0, global-best topology, and the f_1 cost function.

After, the experiments were performed, the obtained information was reviewed and the best results for the four scenarios are presented in **Table 4**. According to these results, using the fourth scenario, the average number of iterations was 1065 in 23.085 minutes to achieve 100 coverage percentage.

Table 5 contains the obtained results for four algorithms: GA, pseudorandom, BPSO, BPSOr, etc. In these experiments, different parameters over the verification platform were changed. In addition, four of the best scenarios are presented showing the best, worst, and average coverage. Also, the average number of iterations and the average time are added.

Commonly, the pseudorandom test generation is used to exercise the device functionality during the functional verification. Reviewing the results, at the start, the coverage percentage was increased very quickly. However, after achieving a coverage threshold percentage, more iterations to increase the coverage were needed. For instance, in the case of the UART-IP core, percentages over 95% were obtained.

According to the results the use of meta-heuristic algorithms to guide the search during the functional verification of digital systems is a good alternative because the behavior areas can

Parameters	**BPSOr scenarios**			
	1	**2**	**3**	**4**
Number of particles	9	6	12	3
Number of neighborhoods	2	2	4	1
ϕ max	4	4	4	4
Topology	G-best	G-best	G-best	G-best
f	f1	f1	f1	f1

Table 3. Configuration parameters of the BPSOr algorithm for four scenarios using the UART-IP core for two sequence solutions.

Scenario	**Number of evaluations**	**Best value**	**Worst value**	**Average**	**Time (min)**
1	2137	100	100	100	46.53
2	1608	100	100	100	37.12
3	2719	100	100	100	60.954
4	1065	100	100	100	23.085

Table 4. Results obtained for four different scenarios using the BPSOr algorithm in the proposed platform with a UART-IP core.

Final values	Binary GA	pseudorandom	BPSOr	BPSO
Best value	100	96.09	100	100
Worst value	99.08	94.53	100	100
Average value	99.87	95.27	100	100
Average evaluations	7353	8000	2137	2194
Average Time (min)	160.46	174.73	46.538	48.755

Table 5. Functional coverage results obtained using genetic algorithms, pseudorandom generation algorithms, PSO and BPSOr to verify a UART-IP core.

be covered very quickly. In the case of genetic algorithms, the population of individuals can evolve by modifying the test sequences to exercise new features of the device. One of the problems is that the guide is based on the evaluations of all population which evolves by means of operators such as mutation, crossover, etc.; when the population size increases, most number of evaluations in each epoch is required; thus, the simulation time is increased. During the functional verification, percentages over 99% were reached using the UART-IP core and the FIFO memory using less time than pseudorandom generation.

On the other hand, when the BPSOr algorithm was used in the verification platform, more functionality was exercised requiring less number of evaluations. Different from the original version of PSO, the BPSOr algorithm can re-initialize the particle swarm based on the current best coverage percentage and the number of iterations performed on run time. It means, if the coverage percentage is not increased, then the best solution, the best particle positions, and the positions and velocities of the particles are reinitialized. This mechanism is used to avoid to fall in local optima solutions and guide the search to behavior regions not explored. In addition, in most of the experiments, the coverage results obtained with BPSOr algorithm were higher than PSO algorithm. It is important to mention that meta-heuristics can be useful techniques to guide the test generation during the verification of devices.

6. Conclusions

Complexity of digital systems is constantly increasing; therefore, the implementation of new methods to improve the confidence and reduce the time of design is required. In this chapter, a verification method based on the use of meta-heuristic algorithms is described. Techniques such as genetic algorithms and particle swarm optimization algorithms were used to verify the digital systems, and a comparison was presented. Also, elements such as coverage models, fitness functions, and software tools are included. According to the results, the use of meta-heuristic algorithms such as the BPSOr algorithm and fitness functions can be useful to exercise the device functionality by focusing on behavior regions that have not been covered. In the case of GA, the coverage results obtained show that a lower number of iterations than pseudorandom test generation is required. Although in the best coverage scenarios a coverage percentage of 100 was obtained, it was observed that when increasing the number of

individuals, the number of iterations used was increased; thus, more time was used in each iteration. The PSO algorithm obtained higher coverage percentages than GA and pseudorandom generation. A main characteristic is that a fewer number of individuals or particles than GA are required. In the case of the BPSOr algorithm, the number of iterations required was less than PSO and GA in most of experiments; therefore, the verification time was reduced. Consequently, hybrid verification methods can improve the performance during the functional verification at block level of digital systems.

Author details

Alfonso Martínez-Cruz[1]*, Ignacio Algredo-Badillo[1], Alejandro Medina-Santiago[1], Kelsey Ramírez-Gutiérrez[1], Prometeo Cortés-Antonio[2], Ricardo Barrón-Fernández[3], René Cumplido-Parra[1] and Kwang-Ting Cheng[4]

*Address all correspondence to: amartinezc@inaoep.mx

1 National Institute of Astrophysics, Optics and Electronics, San Andres Cholula, Mexico

2 Tijuana Institute of Technology, Tijuana, Mexico

3 Computing Research Center, National Polytechnic Institute, Mexico City, Mexico

4 College of Engineering, University of California, Santa Barbara, California, USA

References

[1] Bose M, Shin J, Rudnick EM, Dukes T, Abadir M. A genetic approach to automatic bias generation for biased random instruction generation. In: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546). Vol. 1. 2001. pp. 442-448

[2] Samarah A, Habibi A, Tahar S, Kharma N. Automated coverage directed test generation using a cell-based genetic algorithm. In: 2006 IEEE International High Level Design Validation and Test Workshop. Nov 2006. pp. 19-26

[3] Shen H, Wei W, Chen Y, Chen B, Guo Q. Coverage directed test generation: Godson experience. In: 2008 17th Asian Test Symposium. Nov 2008. pp. 321-326

[4] Li M, Hsiao MS. An ant colony optimization technique for abstraction-guided state justification. In: 2009 International Test Conference. Nov 2009. pp. 1-10

[5] Puri P, Hsiao MS. Fast stimuli generation for design validation of rtl circuits using binary particle swarm optimization. In: 2015 IEEE Computer Society Annual Symposium on VLSI. July 2015. pp. 573-578

[6] Fine S, Ziv A. Coverage directed test generation for functional verification using bayesian networks. In: Proceedings of 2003 Design Automation Conference (IEEE Cat. No.03CH37451). June 2003. pp. 286-291

[7] Cruz AM, FernÃ¡ndez RB, Lozano HM. Automated functional coverage for a digital system based on a binary differential evolution algorithm. In: 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence. Sept 2013. pp. 92-97

[8] Chen W, Wang LC, Bhadra J, Abadir M. Simulation knowledge extraction and reuse in constrained random processor verification. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). May 2013. pp. 1-6

[9] Katz Y, Rimon M, Ziv A, Shaked G. Learning microarchitectural behaviors to improve stimuli generation quality. In: 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC). June 2011. pp. 848-853

[10] Vasudevan S, Sheridan D, Patel S, Tcheng D, Tuohy B, Johnson D. Goldmine: Automatic assertion generation using data mining and static analysis. In: 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010). March 2010. pp. 626-629

[11] Xie T, Mueller W, Letombe F. Mutation-analysis driven functional verification of a soft microprocessor. In: 2012 IEEE International SOC Conference. Sept 2012. pp. 283-288

[12] Cruz AM, Fernández RB, Lozano HM, Ramírez Salinas MA, Villa Vargas LA. Automated functional test generation for digital systems through a compact binary differential evolution algorithm. Journal of Electronic Testing. Aug 2015;**31**(4):361-380

[13] Kennedy J, Eberhart RC. A discrete binary version of the particle swarm algorithm. In: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. Vol. 5. Oct 1997. pp. 4104-4108

IntechOpen