applied sciences

MDPI

Article

Coverage Fulfillment Automation in Hardware Functional Verification Using Genetic Algorithms

Gabriel Mihail Danciu and Alexandru Dinu *

Department of Electronics and Computers, Transilvania University of Brașov, RO-500036 Brașov, Romania; gabriel.danciu@unitbv.ro

* Correspondence: alexandru.dinu@unitbv.ro; Tel.: +40-728-740-857

Abstract: The functional verification process is one of the most expensive steps in integrated circuit manufacturing. Functional coverage is the most important metric in the entire verification process. By running multiple simulations, different situations of DUT functionality can be encountered, and in this way, functional coverage fulfillment can be improved. However, in many cases it is difficult to reach specific functional situations because it is not easy to correlate the required input stimuli with the expected behavior of the digital design. Therefore, both industry and academia seek solutions to automate the generation of stimuli to reach all the functionalities of interest with less human effort and in less time. In this paper, several approaches inspired by genetic algorithms were developed and tested using three different designs. In all situations, the percentage of stimulus sets generated using well-performing genetic algorithms approaches was higher than the values that resulted when random simulations were employed. In addition, in most cases the genetic algorithm approach reached a higher coverage value per test compared to the random simulation outcome. The results confirmed that in many cases genetic algorithms can outperform constrained random generation of stimuli, that is employed in the classical way of doing verification, considering coverage fulfillment level per verification test.

Keywords: genetic algorithms; functional coverage; automation; artificial intelligence; hardware verification

Citation: Danciu, G.M.; Dinu, A. Coverage Fulfillment Automation in Hardware Functional Verification Using Genetic Algorithms. *Appl. Sci.* **2022**, *12*, 1559. https://doi.org/10.3390/app12031559

Academic Editor: Ilaria Bartolini

Received: 15 December 2021
Accepted: 28 January 2022
Published: 31 January 2022

1. Introduction

1.1. The Opportunity of the Present Research

Our society is highly influenced by the trend of digitalization. A continuously increasing number of industry branches are turning their attention to automating their manufacturing process to use robots and automated production lines to strictly monitor the entire production flow and to centralize and digitize business-related data. The same trend influences domestic consumers as well, as they are able to acquire many electronic devices that increase their comfort and ease their daily activities. Furthermore, advances in electronics cause people to frequently change their electronic gadgets in order to benefit from the latest discoveries materializing in new devices. Therefore, manufacturing of electronic devices is highly requested in our society by a wide range of consumers. Consequently, there are many companies which work in different stages of electronics production, and any improvement in the manufacturing flow of these devices can have an important impact over the entire society.

1.2. The Steps in Manufacturing of Integrated Circuits

Most electronic devices are controlled by an integrated circuit (IC), which vary in complexity from the simple ones embedded, for example, into a bicycle safety light to the very complex ones found, for example, in smartphones; these are known as System-on-Chip [1] which either can (in the case of, e.g., microprocessors) or cannot (e.g., Application-Specific Integrated Circuits, or ASICs) be programmed by the user.

Development of an ASIC can last from about a year to several years; in semiconductor factory, the minimum time for manufacturing is about six months. In addition, this process involves tens of persons and, in most cases, several companies. Considering this information, every improvement which might be represented by, for example, automating a step in the logical design or the manufacturing process can save important resources and permit reduction of the time-to-market period of the developed product. In order to provide the reader with a short overview of the manufacturing flow of ICs as well as to introduce the role of functional verification in this process, the principal manufacturing steps of ASICs are enumerated below; see also Figure 1. The blue color indicates relation to the design stage and the green color relation to the foundry stage, which is not in the focus of this paper.

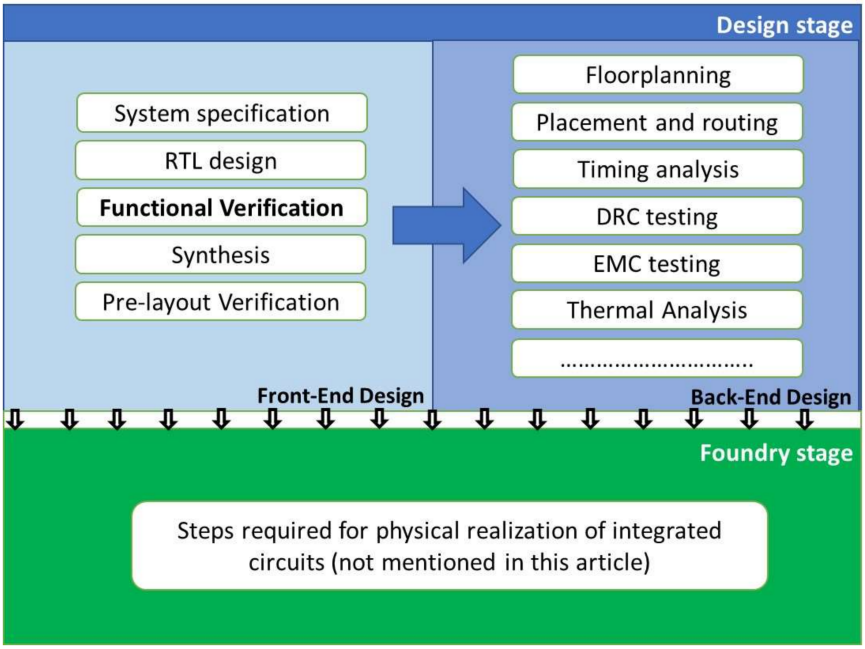


Figure 1. Steps in the manufacturing flow of an integrated circuit.

As observed in Figure 1, manufacturing of ASICs consists of two main parts: design flow (which implies several types of simulations performed involving the design in question) and physical realization of the product (which implies manufacturing of the integrated circuit in the semiconductor factory). The IC design stage can be divided into two parts, front-end and back-end. Front-end design focuses on implementing the functional characteristics of the design according to its specifications, while back-end focuses on the steps required for building a physical implementation of the digital version of the design such as floor planning, placement of logic cells to design a routable chip that meets timing and other constraints, and routing [2]. In this step, multiple aspects such as timing analysis, design rule checks, electromagnetic compatibility (EMC), thermal analysis and other checks are considered. The functional characteristics of the ASICs are mainly implemented in the front-end stage, which implies as principal steps the creation of system specifications, development of Register Transfer Logic (RTL) design, implementation of functional verification, and synthesis and pre-layout verification.

The behavior of a digital design is commonly assessed through conceptual validation of using a physical set-up, e.g., validation with a Field Programmable Gate Array (FPGA) [3].

1.3. Functional Verification: Implementation and Challenges

Functional verification, the application of which is automated in the present work, is one of the most time consuming steps in the manufacturing process of integrated cir-

cuits [4]. This process consists in checking the design functionality (which is implemented through a Hardware Description Language, or HDL) against the specification of the system. Functional verification is implemented in the industry by applying the concepts found in the verification methodologies.

According to [5], there are three types of verification methodologies: dynamic functional verification (the most widespread method), hybrid functional verification and static functional verification. In the current work, dynamic functional verification is used in light of the Universal Verification Methodology (UVM). This methodology, widely employed by verification engineers from all around the world [6–8], was developed by the Universal Verification Methodology Working Group launched by the Accellera organization. This methodology contains guidelines to create a testbench which can be used for functional verification and embeds the experience of the three major players in the IC Verification Industry, which have created tools to support this methodology: Mentor (now acquired by Siemens), Cadence, and Synopsis.

The functional verification process is conducted using metrics, which are the elements that allow verification engineers to measure the progress of verification. By far the most important metric is coverage. Coverage can be performed over a large number of elements which indicate whether the Design Under Test (DUT) contains bugs. The most important coverage metric is functional coverage. Functional coverage is one aspect of the overall coverage analysis process, and contains predefined presumptions and their corresponding expected outcome [9]. In short, functional coverage has in focus all possible cases that might occur when using a circuit. Verification engineers can be confident that a DUT is problem-free if all situations of operation for the respective design are simulated without any errors appearing. Because the currently developed DUTs have many input and output ports, it is not feasible to generate all combinations of input values which could be driven to a DUT [10,11]. Moreover, arranging the same stimuli into different sequences can often bring DUTs into different states. Therefore, generating all combinations between a DUT's inputs does not assure that the design entered all possible operating states. On the contrary, there will be plenty of DUT functionalities that remain uncovered. Fortunately, generating all combinations of stimuli is not needed in order to perform a thorough DUT verification. The number of DUT functionalities is considerably lower than the number of stimuli combinations that the design can read. Coverage elements, which are special data structures supported by UVM methodology, are in charge of registering each different situation met during DUT simulation. Because the DUTs are very complex, combinations between multiple coverage elements are frequently used to describe an operating state of design.

However, it is not easy to bring the DUT into all its possible operating states. For each situation of functionality the DUT needs to be supplied with specific sequences of input values. The more complex the digital design, the more complicated these sequences of stimuli must be. This problem is currently solved by verification engineers using four steps:

1. The functionality of the DUT is split into sub-functionalities; this happens even from the creation of the verification plan at the beginning of the verification process.
2. By analyzing each functionality, the verification engineers understand which inputs of the design contribute to the activation of that functionality. Given the functional specifications, the engineers search for the proper numerical ranges which must fit the values of each input in order to generate appropriate stimuli for achieving the functionality in focus. The numerical ranges found for each input must be inside the legal range according to the functional specification.
3. By applying the constraints found in Step 2, the DUT is supplied with random values through each input relevant for the functionality in focus. By running an appropriate number of simulations, the coverage value related to the design functionality in focus should be as close by 100% as possible. However, in some complex situations this score can be considerably lower.

4. The verification engineers analyze any coverage holes and aim to create specific cases using directed tests to hit each DUT operating situation which was not met in previously-performed simulations. This step represents tedious work and can take a long time. The effort needed for building sequences of input values is consistent because the dataflow required to achieve specific or corner-case DUT functionalities can embed many steps. In addition, this activity requires good knowledge of the design functionality and solid experience in functional verification activities.

In the normal flow of verification activities, the important steps of which are summarized above, two problems can be observed:

i. The large amount of time needed to perform several random simulations to achieve a value of coverage as high as possible with low human effort.
ii. The substantial effort needed to choose the exact values (or small interval of values) for input stimuli to create directed tests that aim to reach the coverage holes existing after the performance of random simulations.

Given these two problems, the functional verification process could be significantly eased if an automatic mechanism for generating the input values needed to activate the desired design functionalities were created. The present paper aims to solve this problem, being part of several initiatives developed worldwide which focus on automating the verification process. However, to the best of the authors' knowledge, until now there has not been a universal solution found that could fully automate the process of finding the correlation between the input stimuli of a DUT and the coverage elements that characterize the DUT's functionalities. Another aim of the current work is to acquire shorter simulations without loss of verification quality. With this achievement, it will be possible to more quickly verify the entire set of DUT functionalities without significant time loss. This action is required each time the DUT is modified due to a change request. Before presenting the current progress in this topic by consulting the literature, it must be mentioned that an appropriate candidate for realizing the correlation–causality effect (in this situation, the causality is represented by the values driven to the DUT inputs and the effect is represented by the fulfillment of coverage for elements in focus) is artificial intelligence. Running multiple simulations and grading the input stimuli based on the coverage level achieved, the input sequences can be modified to better activate a desired functionality of the DUT or to cover all DUT functionalities related to different aspects of design operation. In the current work, the authors propose to automatically modify the initial set of constrained randomly generated sequences until all DUT functionalities in focus are reached. This purpose is achieved using the evolutionary process provided by genetic algorithms. In reading the relevant literature, the authors found that many studies concluded that genetic algorithm-based approaches are the best way of generating sequences of inputs able to reach the desired levels of coverage.

The paper is organized as follows: Section 2 provides information about related approaches of functional coverage fulfillment automation using artificial intelligence techniques, and the main differences between existing studies and the current research are discussed. Section 3 introduces DUTs, verification environments, and the testbench used to validate the approaches currently developed, gives some basic information about the genetic algorithms, and presents the general implementation of the workflow used to automate coverage fulfillment. Section 4 presents the results obtained in each of the three use-cases introduced in current paper. Section 5 discusses the obtained results and emphasizes the key attributes which influenced the performance of the proposed approaches. Finally, Section 6 concludes the paper and introduces a possible direction of future research on this topic.

2. Literature Review

All major industry players and many academic institutions are looking for automated mechanisms of coverage fulfillment, which can shorten the time spent by verification engineers aiming to cover different functional corner-cases. One of the emerging directions

is the automation of the verification process using artificial intelligence techniques. Thus, presents a large number of approaches where the application of functional verification is boosted by transferring a part of the human work to different algorithms running on computers [12]. It can be observed that many parts of the verification process can be automated, such as generation of tests to be used in simulations, generation of components of the verification environments, coverage collection, bug detection, modelling of analog design behavior, and assertions generation. Related to the smart generation of necessary values to be driven at the DUT inputs to reach all operating states of the design [13], collected and summarized many approaches tested on industrial DUTs which demonstrated that the loop between coverage collection and stimuli generation can be successfully closed. The variety of these methods (which can be based on probabilistic methods, data mining, inductive logic programming, etc.) shows that achieving a high functional coverage value related to DUT functionalities in a shorter time represents an important target that justifies extended research activity. Research in [14] shows that between all proposed methods, the application of genetic algorithms is the most lucrative way of influencing stimuli generation to reach a higher coverage value. This conclusion is a driver of the current study, which employs genetic algorithms for reaching the functional coverage fulfillment in a shorter time.

Two relevant papers which aim to increase functional coverage using Genetic Algorithms (GA) are [15] and [16]. In the first-mentioned work, individuals are represented by a set of stimuli generation constraints. By creating multiple generations of individuals and combining the best-performing ones, the aim of this work is to reach a high level of functional coverage. Compared to [15], the current work focuses on generating the values which will be transmitted to the DUT inputs, rather than generating the constraints which should be applied at random generation process. In this way, the current work reduces the randomness effect in the evolution of individuals, aiming to more quickly reach the coverage target. Furthermore, in [15] the genetic algorithm-based approach was implemented in C++. The current work proposes a modern implementation of genetic algorithms using Python language [17], which is supported by a wide range of platforms and which is commonly used in implementation of projects based on artificial intelligence [16]. Another difference between [15] and the current study is that the approaches used in that study were developed in work authored by Samarah et al. and tested over designs modeled in SystemC. In contrast, the currently developed approaches were tested over designs modeled in Verilog which are ready to be synthesized. The benefit of the currently presented approach is that additional efforts for developing a SystemC validation environment can be avoided, if applicable, for the project which benefits from this means of automation. In [18], the individuals consist of sequences of 100 values. They are created and evaluated considering two types of functional coverage items related to differential pulse-code modulation (DPCM). The best-performing sequences of stimuli in terms of coverage fulfillment are further driven to the inputs of both the DUT and its reference model. The target of individuals is to make DUT outputs reach a value higher than 50 or a value lower than -50 for a predefined number of times. GA succeeded in generating a proper set of data for ensuring the coverage fulfillment. Similar to [15], in the current work only a DUT input is driven with data generated using genetic algorithms. However, in the case of the present paper most individuals contain only 20 values, their size being considerably smaller compared to the individuals developed in [18], which contained 100 values. Despite this difference, the approaches currently developed reached the maximum coverage score several times. This paper [19] can be considered a continuation of the work in [18]. One of the targets of the mentioned work was to reach the same functional coverage items many times. To reach this aim, multiple stimuli generation methods were tried, such as roulette-wheel and tournament. Following their, in the current work various approaches based on genetic algorithms were created and fine-tuned. An important difference between [19] and the current work is that in [19] each individual consisted of values needed by multiple inputs

of the DUT. In all three examples analyzed in the current work, the individuals are used to build sequences of values to be driven at the same input of the DUT.

Several studies have aimed to accelerate functional coverage fulfillment without using genetic algorithms. For example, in [20], supervised learning was used for training models able to realize a correlation between DUT input stimuli and each coverage element (called "coverage bin"). After running many simulations and extracting information from generated reports, the author succeeded in reaching the coverage target by parsing all coverage elements and generating, for each of them, the required stimuli to bring the DUT to the desired state. In the current work, by using the genetic algorithms, the learning phase necessary in the supervised learning approach could be skipped, thus shortening the time required for the verification process. However, in the present work, as in [20], in order to increase the efficiency of the training process the coverage elements are implemented in Python, not in SystemVerilog.

The general approach based on genetic algorithms developed in this work is similar to the one proposed by Subedha et al. [21]. The most important differences are the proposed data and scenarios, as well as the parameter adjustments. In [21], the coverage elements focus on the functionality of the software systems; however, in the current work the verification of hardware designs has be automated. However, as both systems consist in defining a logical order of operations to achieve a well-described functionality, the developed approaches share many common points.

Using genetic algorithms, the current work aims to build high-performance verification tests which can obtain a high coverage value for the DUT functionality in focus using a limited number of stimuli. In addition, the we aimed to create multiple best performing sets of stimuli for each coverage target. By running them, they uncovered hidden problems more quickly, usually correlated with functionality corner-cases of design. In this work, the we demonstrate the efficiency of genetic algorithm approaches over constrained random verification, considering the above-mentioned aspects. The originality and the novelty of the article consist in the development of different implementations of genetic algorithms. The various methods of generation of children based on their parents within genetic algorithm approaches, the construction of following generations inheriting members from previous generations, and the examples used for validating the efficiency of the methods proposed in the current work represent original ideas that are further described in this paper.

3. Materials and Methods

3.1. Employed DUTs and Coverage Targets

To assess the approaches developed under the current research, the team used three testbenches consisting of one DUT and its corresponding Verification Environment (VE).

The first design implements an Arithmetic Logic Unit (ALU) which can perform the four basic mathematical operations. It is used as an accumulator; in addition to the first operation, its first operand always copies the result of the previous operation. During simulations, the design receives the following data: the first value of the first operand, 20 values of the second operand (in each simulation, 20 operations are performed), and 20 operation codes (one code is assigned to each of the four basic operations). At each operation, the ALU outputs the result of the operation (when calculation trigger arrives) and an error bit for overflow and underflow situations. In experiments performed during this study, both the operands and the result were 8 bits wide, as shown in Figure 2.

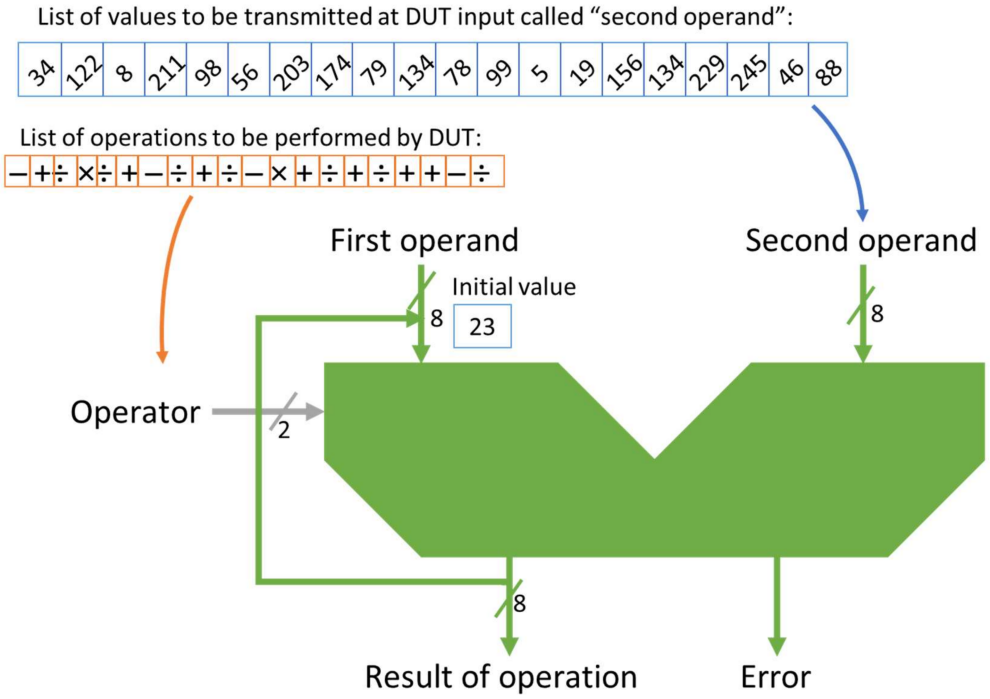


Figure 2. The representation of a DUT representing the Arithmetic Logic Unit and of an example of the data that the ALU receives during a simulation.

In Figure 2, all data contained by an individual are represented. The first operand input receives the initial value from the individual, and the other values are received from the output of the DUT. The second operand input receives all 20 values from the individual. The operator receives the codes of the operations to be performed (2 bits are enough to represent the four basic mathematical operations).

The range of values at the output of DUT is split into 10, 15 or 20 intervals. The target of the implemented automation process consists of stimuli generation to obtain at least one result in each of these intervals during the 20 operations performed.

The second design represents a smart lamp controller that can turn off a light bulb or light a bulb at three different intensities: low light intensity, medium intensity and high intensity. In [22], a device similar to the one verified in this paper is presented. These intensities are controlled by a sensor or by receiving commands from a button. The readings from the sensor are 8 bits wide, meaning 256 different outcome values. The target is to monitor the values coming from the sensor and, similarly to the previous DUT, to count the number of intervals in the available range of values for which at least a sensor value was generated. Similarly with the previously described situation, in each test, 20 data transactions were performed.

The third design implements an Ambiental controller receiving values from three sensors: luminosity, humidity, and temperature. Such an electronic device finds its place in any smart house [23]. Each sample from the input luminosity signal is 10 bits wide (the maximum accepted value is 1024), although the outcome values of the controller are limited to the 0–900 range. Ambiental controller operation can be enabled or disabled by pressing a latching button. The coverage target consists in generating at least one value inside all defined ranges of the luminosity signal. To achieve this, a sequence that emulates a sensor performing 20 transmissions is employed. The second coverage target is to cover all defined ranges of the luminosity signal only when module operation is enabled.

3.2. Verification Environments

The Verification Environments (VE) followed the Universal Verification Methodology (UVM) [24] and were written in SystemVerilog language [25]. The verification environment

for DUT, consisting of an accumulator, does not inherit components from UVM, although it is designed using UVM concepts. The verification environments built for the smart lamp controller and Ambiental light controller inherit components from the UVM library. As represented in Figure 3, stimuli generated in Python by genetic algorithms are written in text files. These files are read by sequences from the verification environment during verification tests. In addition, the tests run with ModelSim simulator [26] are started from the Python environment. The software script can configure the verification environment using parameters. The parameters are defined in VE top file and are propagated to any components using them. Finally, the results of simulations are extracted from simulation logs and are read by Python programs which will use them to generate the stimuli for the next generations.

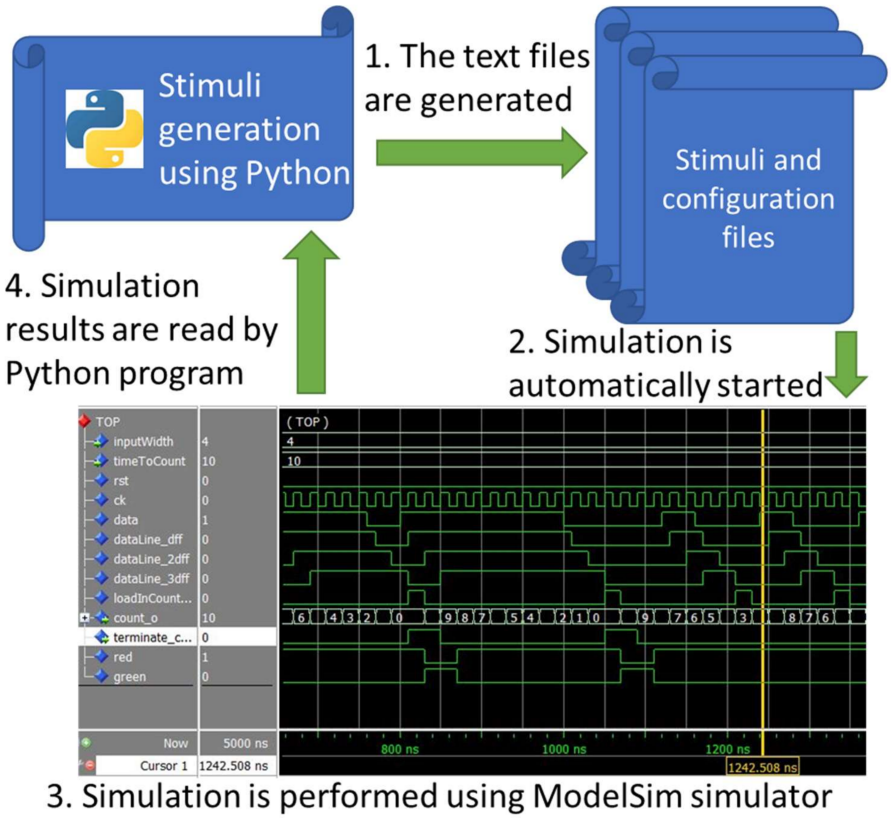


Figure 3. The information processing flow used in the current work.

A similar system of information transfer between Python program and verification environment can be consulted in [27].

3.3. Description of General Approach (Entire Environment)

The implementation of the genetic algorithms was carried out in Python, the simulation testbench was written in SystemVerilog, and the DUT was written in Verilog language.

The monitors contained by verification agents collect signal values from interfaces and send them to coverage collectors to be registered. Additionally, certain values are collected directly from scoreboards to check coverage fulfillment for the functionalities of different DUTs. The value of coverage in focus is printed out at the end of each ModelSim simulation. After the end of each simulation, the Python environment reads the simulation report and extracts the coverage value. Thus, a correlation between input stimuli and their outcome is obtained.

3.4. Operation of Genetic Algorithms

Use of genetic algorithms is considered one of the best practices for speeding up functional coverage fulfillment [13]. Two key attributes of these algorithms are that they "ensure global optimization and have a high degree of parallelism" [14].

Using genetic algorithms, multiple stimulus sets are combined in a close-to-optimal way. If two stimuli sequences provide a good coverage value, it can be considered that a combination between them will be beneficial as well. Therefore, when combining these lists of stimuli in different ways, other datasets are generated which have the potential to improve the coverage fulfillment level. As genetic algorithms provide approximated solutions close to the optimal one, more than one set of stimuli can be obtained as potential results for a desired coverage.

Genetic algorithms begin with the generation of a random population of stimuli. A population consists of individuals. Each individual contains a sequence of stimuli which will be further driven to the DUT using the verification components usually employed according to the UVM methodology (sequence, sequencer, and driver). In Figure 4, the representation of an individual which contains values 8 bits wide can be seen. These values are transmitted to the DUT input called the "first operand". In some implementations, multiple groups of individuals are generated for the same verification environment. Data from each group are transmitted through a specific interface signal to the DUT.

43	63	188	1	255	19	202	4	7	55	99	12	178	239	43	134	68	33	142	20

Figure 4. The representation of an individual containing 20 values of data items, each 8 bits wide.

There are two main operators which are used in most implementations of genetic algorithms in the industry, namely, crossover and mutation [28,29].

Crossover operators combine two individuals by breaking them into two parts at a random position and switching their second part, as represented in Figure 5.

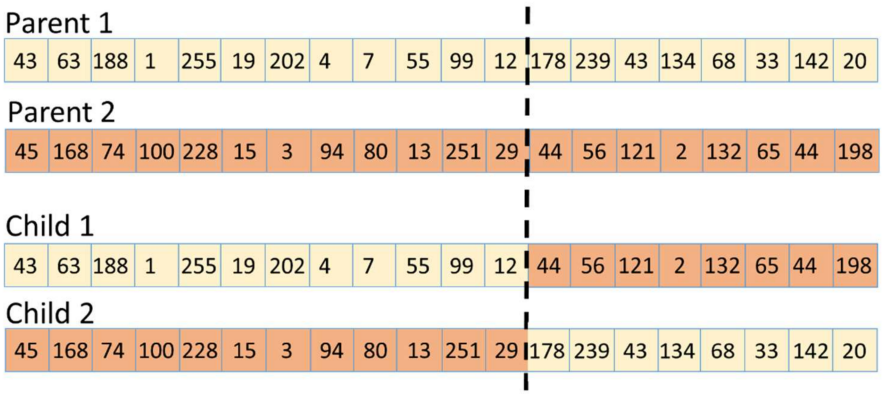


Figure 5. Crossover process representation.

The initial individuals are called parents, and the individuals resulting from combinations of parents are called children [30]. The crossover process is mathematically described for the first child using (1). The first child takes the genes of parent 1 until the breaking point and the genes of parent 2 found after the breaking point. For the second child, the complementary behavior can be easily deduced.

$$child_1[i] = \begin{cases} parent_1\,[i],\ if\ i \leq random\_number \\ parent_2[i],\ if\ i > random\_number \end{cases}, i \in [1,2] \quad (1)$$

where *random_number* is a randomly generated value in the range of the number of bits/elements of the stimuli list. The *random_number* represents the crossing point (the dashed line in Figure 5). In this way, if two parents obtained good performance in terms

of functional coverage fulfillment, the children embedding the "best part" of each parent could give even better results. Furthermore, two parents who initially were not valuable in terms of coverage fulfillment could generate high-performing children by creating other combinations of input stimuli. If more breaking points are employed, the same parents can generate more than two children.

The mutation operator is applied by changing a few of their values (randomly selected); in the current implementation, one or two values were altered inside the structure of children. This helps the algorithm to avoid local maxima, represented by high coverage values the population converges to which are not the best overall score. The effect of the mutation operator is represented in Figure 6.

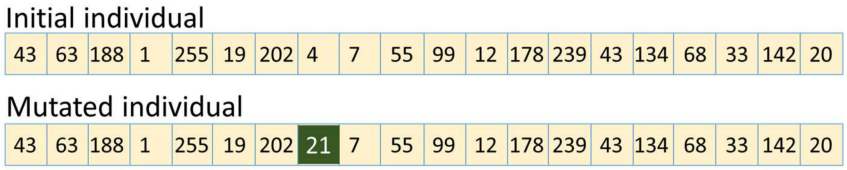


Figure 6. Mutation operator representation.

In Figure 6, the element with value 4 and index 8 was mutated. In this way, a fresh offspring appeared in the available population of individuals. Considering the information above, the number of children can exceed the number of parents. Therefore, after creating all of the desired children, a selection process is applied, as can be observed in (2). Therefore, only the best individuals are retained in the selected population to participate in the genetic operations (cross-over and mutation).

$$SP = \{x[i] : x[i] \in IP, \forall\, i < n_{pop}\} \quad (2)$$

where SP represents the selected population (whose elements will perform cross-over process), IP represents the initial population generated in the previous iteration (which now has elements arranged in descending order), and n_{pop} represents the basic number of a population. First, each child is evaluated individually during a ModelSim simulation and the obtained value of coverage in focus is correlated with the individual which generated it. Secondly, a few of the best performing parents (which thus survive in the next generation; in the current situation, 20% of parents survived) and all obtained children are ordered in descending order considering their coverage value. In the end, a limited number of individuals (which can be equal to the number of parents from the previous generation) become the parents in the new generation, as shown in (2). In this way, it is expected that the population evolves from one generation to another, and that in the end the best individuals will obtain the maximum possible coverage value.

3.5. Description of Developed Programming Ecosystem Based on Genetic Algorithms

Development of all functions entitled to implement the genetic algorithm focuses on creating a programming ecosystem that can deliver good results as quickly as possible. In the first instance, the coverage targets are stated using different parameters. The most important parameters used in the current work are the number of intervals in a signal value range that should be hit, the number of generated transfers, the width of signals to be covered, the number of generations (or iterations), the number of parents in each generation, and the number (percentage) of individuals that converge to the desired outcome.

The steps in the application of genetic algorithms in the current work are depicted in Algorithm 1.

Algorithm 1 Genetic Algorithm

```
Initialize population with random values in accepted range
Initialize algorithms parameters:
    n_iter                  number of iterations
    no_of_operations        number of operations which are executed before coverage is collected
    r_mut                   mutation probability
    width                   width of operators
    n_bits                  no_of_operations * width
    pop                     population existing into a generation
    n_pop                   number of individuals in a population
    r_cross                 crossover rate (20% of best genes are directly copied as future parents
    no_of_coverage_intervals  number of coverage intervals for result value
    best_eval               score for the best evaluated individual, initially 0
Initialize pop with random generated individuals
repeat n_iter times
        Evaluate each individual using OBJECTIVE → array named scores is obtained
        for i in [1 .. n_pop]
            if scores[i]> = best_eval
                                        best_eval = scores[i]
                                        insert individual in sorted array named best
        select best n_pop/2 performing individuals from scores → array named selected
        sort selected array in descending order based on the evaluation score
        for i in [1 .. len(selected)]
            p1, p2 = selected[i], selected[I + 1]
            if (i<1-r_cross)g*n_pop
                                        children.append(p1)
            apply crossover operator between p1, p2 →the children c1, c2 are obtained
            if (random < r_mut)
                                        apply mutation on c1 or c2
            children.append(c1,c2)
        pop = children
return best
```

The general description of the objective function, which represents the fitness function in the current approach, is found in Algorithm 2:

Algorithm 2: Objective

```
Input:
        x                       Individual representation as an array
        width                   the width (measured in bits) for each gene in individual
        path_to_sim             Path to folder where the simulator is located
        no_of_coverage_intervals  Number of desired coverage intervals in which the values of data of interest is split
Output: the coverage value for x
Using x data, generate a stimuli file in which the values of interest (i.e., luminosity values) are prepared to be used in the next step: simulation
Using input parameters, call a subprocess that simulates the DUT behavior in ModelSim
Extract coverage values from calling the simulation report
return coverage value
```

For better understanding, the steps for applying the genetic algorithms seen in Algorithm 1 are further explained here. First, the initial population is randomly generated,

containing a fixed number of elements (the population size). Each population contains a sequence of data items representing the stimuli for each performed transaction during a simulation, as represented in (3). The elements can be represented either as binary or as decimal numbers (the researchers used both options during the current work). Because inside each generation there will be the same number of parents, each of them having the same structure, the next part of the program is common for all generations (represented by consecutive program iterations).

$$X = \{x \in \mathbb{N} : min \leq x \leq max\},\ |X| = NOOPS \tag{3}$$

where *min* and *max* represent boundaries of legal values range for each individual (i.e., 0–900 is the legal range of luminosity values employed in case of Ambiental controller), and *NOOPS* is the abbreviation for "Number of Operations Per Simulation" and represents the population size. As an exception, in the case of ALU the individuals contain the values of the operands B, the initial value of operand A, and the operator codes for each operation performed. In the case of a smart lamp controller and an Ambiental controller, each individual contains only a fixed number of values provided by only one sensor.

During each generation, the score for each individual is computed. This means that a ModelSim simulation is started for each individual where the DUT is provided with corresponding stimuli. At the end of the simulation, the obtained coverage value is associated with the individual used for running the simulation. This coverage value represents the fitness function cost. Further, the stimuli provided by the best performing individual (there can be several individuals who obtained the highest score until that moment) are saved on disk, representing a precious output of the programming ecosystem. As a key element in this work, the best performing individuals are stored into a list in order to avoid saving the same set of stimuli multiple times. The concept used for saving the best individuals in each generation on disk is mathematically represented in (4):

$$SE = \{I \in CG : score(I) = \text{max\_score},\ I \notin BIL\} \tag{4}$$

where *SE* (Saved Element) represents, one by one, each element saved onto disk, *I* represents each individual in the current generation, *CG* (Current Generation) represents the set of individuals from the current generation, *max_score* represents maximum coverage value reached at computation moment by all individuals, and *BIL* (Best Individuals' List) represents the list created by Python program which contains all individuals having the maximum score at the time they were generated. Therefore, after calculation of *SE*, *SE* will be copied to *BIL*.

After each individual is associated with its coverage value, the individuals are ordered in descending according to their scores.

When creating the new population, the best performing individuals from the previous generation are always kept (a configuration parameter dictates how many parents are copied to the following generation) in order to allow the algorithm to converge.

Multiple ways to select the combinations of parents presented were tried during this study. A comparison of these approaches is provided in the results section. After the crossover process finishes, the mutation operator is applied over randomly selected children. During the current study, there were cases when children were not saved if they were identical to their parents. In this situation, if at the end of the crossover operations over all parents the number of children is less than the initial number of elements in the population, other individuals are randomly generated.

At the end of each iteration, the children become the parents of the next generation, and the steps described above are reloaded.

4. Results

During the current work, several approaches based on genetic algorithms were developed in an attempt to achieve the best results with the minimum processing time. Below,

differences between different employed methods of automating the coverage fulfillment are presented for each DUT along with the results obtained.

If not explicitly mentioned, each generation (the number of parents per generation) for the above-mentioned trials contains 20 individuals (n_pop = 20).

4.1. Coverage Collection for ALU Operating as an Accumulator

The design represents an arithmetic logic unit (ALU) operating as an accumulator (the first operand is fed with the value of the previous operation result), which performs the four basic mathematical operations. The operands (called operand A and operand B) and the result are 8 bits wide, and there is an error bit that signals both overflow and underflow situations. The range of the resulting values was split into 10, 15, and 20 ranges during the many trials performed. All coverage ranges contained the same number of values, except for the last group which accommodated only the remaining values, as depicted in (5). The coverage target consists in obtaining results from the DUT, the values of which fit each of the intervals computed in (5) after 20 consecutive operations were performed.

$$ULs = \left\{ c \in \mathbb{N},\ c = \max\left(bottom + i + \frac{top - bottom}{n}, top \right) | i \in (0, n-1) \right\}, \quad (5)$$

where *ULs* (upper limits) represents the list of the upper limits of the ranges of values corresponding to the intervals of values which must be covered (the list of lower limits can be easily deduced as the coverage intervals are arranged back-to-back); *top* and *bottom* represent a maximum and minimum value which can be reached by the signal to be covered (e.g., if the signal is 8 bits wide, and there are no other constraints imposed by functional specification, the top value is 255 and the bottom value is 0).

The approaches used to generate sequences of stimuli are labeled and their characteristics described in Table 1.

Table 1. The characteristics of genetic algorithm-based approaches developed in the present work.

Version of GA	Characteristics of Genetic Algorithms-Based Approach
Version 1	• the data contained by each individual which commands the accumulator to perform a sequence of 20 operations are stored in this order: the first value of operand A, the first value of operand B, the first code of operator, the second value of operand B, the second code of operator, . . . the 20th value of operand B, the 20th code of operator; • the crossover is done using a random position in the data stream that represents an individual; • two parents generate exactly two children; • only the best parents (the first half of the total number of parents which are extracted after ordering the individuals) are used to generate children in the next population (parent 1 is combined with parent 2, parent 2 is combined with parent 3, . . . , parent n_pop/2-1 is combined with parent n_pop/2); • 20% of the parents (randomly selected) are transferred unaltered to the next generation; • mutation probability is 2%.
Version 1.1 (compared with v1.1)	• all parents are used to generate the next population (parent 1 is combined with parent 2, parent 3 is compared with parent 4, . . . , parent n_pop-1 is combined with parent n_pop).
Version 1.2 (compared with v1.1)	• the best 20% of parents are copied as children; the parents copied as children are not randomly selected as in previous versions.

Table 1. *Cont.*

Version of GA	Characteristics of Genetic Algorithms-Based Approach
Version 2 (compared with v1)	• the data contained by each individual are stored in this order: the first value of operand A, the first value of operand B, the second value of operand B, ..., the 20th value of operand B, the first code of operator, the second code of operator, ... The 20th code of operator. • the best 20% of parents are copied as children, then these parents are not recombined anymore • the crossover is done using two breaking points: one breaking point is set to split the values of operands B into two datasets, and one breaking point is set to split the values of the operators into two datasets, as can be seen in Figure 7. Thus, many children are generated by combining these datasets.
Version 2.1 (compared with v2)	• the individuals copied from one generation to another are combined and generate children
Version 3 (compared with v2.1)	• this version is identical with v2.1; the code was not changed in functionality, only modified for better control using parameters
Version 3.05 (compared with v3)	• the best parents are not copied as children; instead, the children are obtained only by crossover, and can be modified by mutation.
Version 3.1 (compared with v3)	• now. the parents are combined in random order (until now, parent 1 was combined with parent 2, parent 2 was combined with parent 3, etc.)
Version 3.2 (compared with v3)	• the children who are identical with one of their parents are ignored; therefore, two parents can have no children; if after all crossover processes the minimum number of children (the size of population) is less than n_pop, new random individuals are generated
Version 3.3 (compared with v3)	• mutation coefficient is not 2% anymore; rather, it increases proportionally with iterations from 0 to 1

Figure 7. Breaking points (sometimes called crossover points) in the structure of individuals from v2.

The described approaches were tested over multiple coverage targets to check their performance. The obtained results can be seen in Tables 2–4, and are grouped by the number of ranges into which the result was split. To emphasize the performance of genetic algorithm approaches over the classical verification technique (known as "constrained-random verification"), the team ran several sets of simulations using randomly generated stimuli. They used multiple metrics to compare the performance of employed approaches:

Table 2. Results for ALU when values of result were split in ten ranges.

Version of GAApproach	Index [2] of First Individual Having 100% Coverage	Index of Generation with at Least 50% of all Individuals Having Maximum Coverage	Index of Generation Having Maximum Coverage in all Cases
v1	19	n/a [1]	n/a
v1.1	62	n/a	n/a
v1.2	252	n/a	n/a
v2	101	6	7
v2.1	25	3	3
set 1 of random generations	19	n/a	n/a
set 2 of random generations	15	n/a	n/a

[1] Situations that were not possible to reach under the corresponding approach are marked in tables with n/a.; [2] The index of each individual is calculated considering the generation the individual belongs to and the index of the individual in the respective generation.

Table 3. Results for ALU when values of result were split in 15 ranges.

Version of Genetic Algorithms Approach	Obtained Results
v3	100% coverage was obtained (beginning with 13th generation) by a stimulus set
v3.05	93.3% coverage was obtained (beginning with 21st generation) by a stimulus set
v3.2	100% coverage was obtained (at 4th and 5th generation) by two stimulus sets
v3.3	93.3% coverage was obtained (at 8th generation) by a stimulus set
random generations	86.7% coverage was obtained by only four items in a population of 400 individuals

Table 4. Results for ALU when values of result were split in 20 ranges.

Version of Genetic Algorithms Approach	Obtained Results
v3	90% coverage was obtained, even after waiting 40 generations to evolve
v3.05	85% coverage was obtained, even after waiting 35 generations to evolve
v3.1	80% coverage was obtained at the 37th generation
v3.2	85% coverage was obtained (at 4th generation), even after waiting 40 generations to evolve
v3.2	90% coverage was obtained when there were 100 individuals/generation from the 16th generation
v3.3	90% coverage was obtained (at the 11th generation) by a set of stimuli
random generations	85% coverage was reached by only a sequence of stimuli in a population of 400 individuals

- Index of the first individual (first set of stimuli) to obtain the maximum coverage value
- Index of the generation (iteration) when at least 50% of parents obtained the maximum coverage value
- Index of the generation when all individuals obtained the maximum coverage value; as in all cases it is possible that multiple identical parents exist, the percentage of identical parents was diminished in approach v3.2

Situations that were not reachable under the corresponding approach are marked in tables with n/a.

When the values range of the result was split into ten parts, the coverage target was reached easily using constrained verification (random generations), as well. The advantage of a genetic algorithm can be considered as the convergence reached by v2 and v2.1 to the global maximum (100% coverage fulfillment for all individuals after 3 or 7 generations). However, these program versions allow multiple identical parents to be generated, and only a few unique stimuli sequences may have been obtained.

When the values range of the operation result was split into 15 intervals, the results shown in Table 3 were obtained. In this case, considering that v3.2 had the best performance, ignoring children which were identical to their parents proved to be very beneficial. An important observation is that all approaches based on genetic algorithms outperformed the coverage fulfillment score reached when only random generations were performed.

However, considering the results from Table 4, where the values range of result was split into 20 intervals, ignoring children identical with parents is not a universal best solution, and randomness in generating the initial individuals remains highly influential in the performance of the genetic algorithm approaches employed. The fact that 100% coverage was not reached is explicable; 100% coverage means that each performed operation hits another result interval, which is very difficult to achieve, thus, 90% coverage means that only 18 different intervals were hit, which represents a very good result.

In this case, only the v3 and v3.3 program versions outperformed the score of the series of random generations, and v3.2 only did so after running thousands of simulations (the case when only 100 parents/generation were kept, although the number of children can be up to six times larger), exceeding the score reached by strategy emulating constrained random generation.

4.2. Coverage Collection for Smart Lamp Controller

The design represents a lamp controller. The lamp has four levels of luminosity: off, low, medium, and high intensity. The lamp can operate in two modes: automatic, when it switches its level of luminosity based on values received from a luminosity sensor, or manual, when pressing a button moves the lamp between the four possible light states. In this use case, the lamp was used only in automatic mode, where its luminosity is influenced only by the values supplied by the light sensor. The signal coverage was in focus, the range of signal possible values being split into 15 or 20 intervals.

Unlike in previous cases, for individuals containing values from the sensor, the crossover was accomplished using only one breaking point. The important differences between the versions of genetic algorithm approaches which were developed are presented in Table 5; the number of versions was chosen to correspond with the approaches used for previous DUT.

Table 5. The characteristics of the genetic algorithm-based approaches developed during the present work related to the smart lamp controller DUT.

Version of GA	Characteristics of Genetic Algorithms-Based Approach
Version 3.2	• all children identical with their parents are ignored; • the crossover is performed between parent 1 and parent 2, parent 2 and parent 3, parent 3 and parent 4, etc.
Version 3.3	• the children identical with their parents are not ignored; • the crossover is performed between parent 1 and parent 2, parent 2 and parent 3, parent 3 and parent 4, etc.
Version 3.4	• the children identical with their parents are not ignored; • the crossover is performed between parent 1 and parent N, parent 2 and parent N-1, parent 3 and parent N-2, etc.
Version 3.5	• the children identical with their parents are not ignored; • the crossover is performed between parents which are randomly selected

By generating 20 random values 8 bits wide and applying multiple genetic algorithm approaches, the results from Table 6 were obtained. For evaluation reasons, the constraint random verification methodology was applied by running 400 simulations with

random stimuli. Most genetic algorithm approaches outperformed the scores obtained by random simulations.

Table 6. Results obtained when the smart lamp controller was stimulated with values provided by programs based on genetic algorithms.

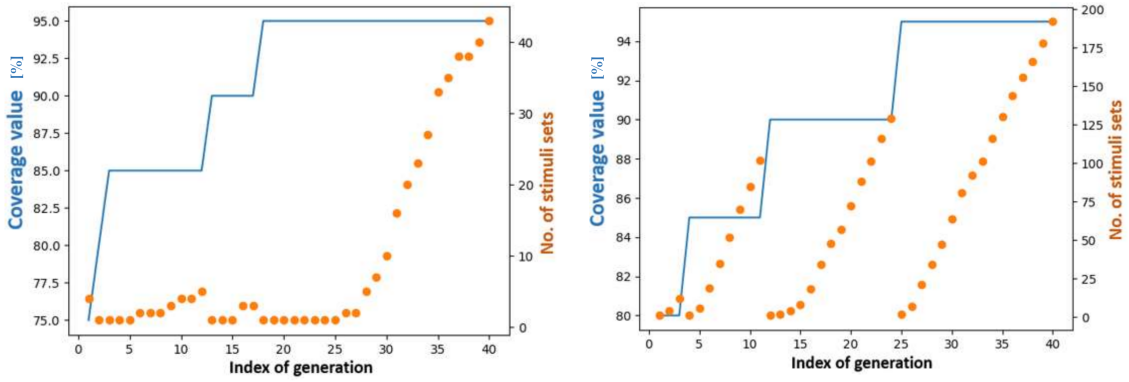
Criterion	15 Intervals					20 Intervals				
	v3.2	**v3.3**	**v3.4**	**v3.5**	**Random Stimuli** [5]	**v3.2**	**v3.3**	**v3.3**	**v3.5**	**Random Stimuli** [5]
maximum reached coverage [%]	100	100	100	100	93.3	90	95	100	85	85
Parents/generation	20	20	20	20	400	20	20	100	20	400
first iteration containing a dataset which leads to maximum coverage	12	4	9	5	1	7	18	9	4	1
first generation when at least n_pop/2 elements having maximum coverage were obtained	n/a	15	22	13	n/a	n/a	28	21	n/a	n/a
number of stimulus sets achieving maximum coverage	1	144 [2]	60 [1]	11 [1]	7	2 [1]	43 [2]	173 [2]	13 [3]	3

[1] all data sets were obtained by mutating the same root stimuli dataset; [2] in addition to one set of stimuli (the one which was first generated), all data sets are based on the same root sequence; [3] the sets were based on four different datasets; [5] equivalent to constrained-random generation of stimuli.

Considering Table 6, it can be seen that most of the approaches based on genetic algorithms had good performance (over 85% coverage). v3.3 obtained the best scores in terms of how quickly the element was reached with the highest coverage score and how many different data sets were generated, even though the population which counts 20 parents was used. If the number of parents was increased to 100, then 100% coverage value was reached. The large number of "winning" stimulus sets appeared due to mutation mechanisms (only one or two elements are different between data sets). In the last row of Table 6 the number of root stimulus sets appears (the base roots, which generated many mutations).

Because v3.4 achieved the worst performance compared with the other genetic algorithm-based approaches when values of the result were split into 15 intervals, it was not tested anymore when 20 intervals were used. The best-performing variants were v3.2 and v3.3.

Figure 8 presents the evolution of the maximum coverage level reached by each generation. This picture shows the advantage of genetic algorithms; by performing additional combinations between individuals iteratively, a higher coverage value is obtained. The individual having the highest score is retained in the next generation. Thus, in a successful simulation, the individuals evolve over time until reaching the maximum possible coverage value. In the current case, 95% coverage was the highest score; when the population had only 20 individuals, this performance was achieved by the v3.3 approach. Orange dots represent the number of sequences that can be used to reach maximum coverage level at one time. Each time a new stimulus set obtains a high score, this number is reset to "1". For example, although both examples reached the same maximum value of coverage, it can be considered that the version from Figure 8b outperformed the one presented in Figure 8a. The reason for this is that the first-mentioned approach discovered almost 190 stimulus sets for maximum coverage value, while the situation in Figure 8a supplied less than 50 high-performing stimulus sets.



(**a**) (**b**)

Figure 8. Two examples for evolution of a population of 20 individuals across 40 generations during the v3.3 approach: (**a**) represents the visual description of the numerical data inTable 6; (**b**) represents the visual description of another result of the same v3.3 approach.

4.3. Coverage Collector for Ambiental Controller

The DUT which implements the Ambiental controller functionality receives input values from humidity, light, and temperature sensors. Based on the received values, the design controls several actuators which are common in a smart house environment (A/C, dehumidifier, blinds). The signal for which the coverage is measured is used to supply the DUT with the values from the luminosity sensor. The legal luminosity values are constrained between 0 and 900 lumens/squared meter. As in the previously described cases, the values range is split into 15 or 20 intervals. The current design reuses the approaches developed for the smart lamp controller. The main difference between the two approaches is that in the case of the Ambiental controller, although the signal is 10 bits wide, the values from 901 . . . 1023 are illegal.

As can be observed in Table 7, only the v3.2 and v3.3 genetic algorithm approaches were tested for the current DUT, as these versions performed best in the case of the smart lamp controller.

Table 7. Results obtained when the Ambiental controller was stimulated with values provided by programs based on genetic algorithms.

Criterion	15 Intervals			20 Intervals			
	v3.2	v3.3	Random Stimuli [2]	v3.2	v3.2	v3.3	Random Stimuli [2]
maximum reached coverage [%]	100	100	93.3	90	95	100	85
Parents/generation	20	20	400	20	200	20	400
first iteration containing a dataset which leads to maximum coverage	12	9	1	13	3	29	1
first generation when at least n_pop [3]/2 elements having maximum coverage were obtained	23	14	n/a	n/a	n/a	35	n/a
number of stimulus sets achieving maximum coverage	10 [1]	128 [1]	15	1	2	30 [1]	1

[1] all data sets were obtained by mutating the same root stimuli dataset; [2] equivalent to constrained-random generation of stimuli.; [3] n_pop represents the initial size of population, which remains the reference number for computing the number of parents during each generation.

Considering the results in Table 7, v3.3 again obtained the best results in terms of coverage fulfillment. Although for v3.2 a trial with 200 parents per generation was run, when the values range was split into 20 intervals the 100% coverage value was not reached. However, as expected, having multiple individuals per generation was beneficial, as the coverage value increased from 90 to 95. As can be seen in Table 7, the approaches based on genetic algorithms outperformed random simulations depicting constrained-random verification. In the case of random simulations, even if 400 sequences of stimuli were

transmitted to the DUT, none of them reached the maximum coverage level achieved by genetic algorithm-based approaches. v3.3 even succeeded in reaching a 100% coverage target when the value of the result was split into 20 intervals; this means that each stimulus in the 20 element sequence reached a different interval of values.

In Figure 9, it can be seen that the evolution of the best-performing approach was v3.3 when values were grouped in 20 intervals. After obtaining the first stimulus set with 100% coverage, there were many generations when no other winning sequence was found. However, by increasing mutation coverage more and more, similar data sets began to appear.

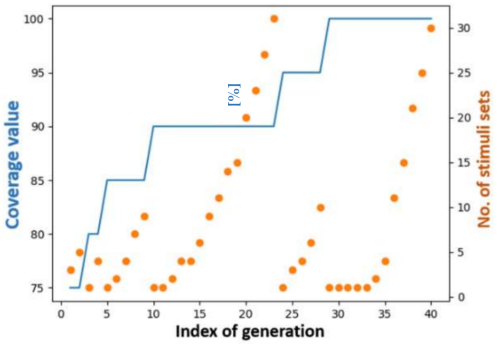


Figure 9. Representation of coverage fulfillment process using best performing approach, v3.3.

Furthermore, in Figure 9 the orange dots show a close-to-ideal evolution process. Even in the situations where the coverage level does not increase, more and more sets of stimuli which achieve the maximum meet the coverage level. When the coverage remained at 90% over more than ten generations, the number of stimulus sets generated with a coverage value of 90% exceeded 30 items.

5. Discussion

Genetic algorithms represent a welcome means of automation for the verification process, if properly configured. During the current work, multiple approaches using genetic algorithms for automation of coverage collection were tested. In this way, important conclusions about the proper configuration of methods used to correlate the coverage value with the input stimuli and about the use of these correlations for accelerating the coverage fulfillment were found. Before presenting our observations, it must be stated that one of the important contributions of the authors of this study is the way in which genetic operations were performed to create several efficient evolution processes. According to Algorithm 1, the fitness function for each individual is represented by the results of ModelSim® simulations. This aspect allows the currently developed approach to be successfully used even in more complex cases when the DUT cannot be completely emulated. In this case, because a complete reference model cannot be created in a software-based system, the only way to obtain details about DUT operation is to use its RTL code.

As a first general observation related to the measurable results obtained in the current paper, it was demonstrated that it is beneficial to propagate the best parents from the previous generations to the next generation and to use them in the cross-over process. A relevant example is the performance of v2.1 over v2, as seen in Table 2. By using the best individuals of each generation in the cross-over process, the first individual having 100% coverage was obtained in about 25% of time compared to the situation when the best individuals were not use for obtaining new children.

In addition, it was demonstrated that the convergence of the genetic-based approaches to a high coverage value is possible only if the best parents are propagated in the next generation. Otherwise, even if high coverage values are obtained after applying genetic operators (cross-over and mutation), these have a chance to be lost from one generation to another. Relevant examples include the outcomes of the v3.05 approach noted in

Tables 3 and 4, where lower maximum coverage values were obtained (93.3% and 85%) compared to the other approaches.

The opportunity of employing genetic algorithms depends on the situation. For easily-reachable targets, constrained random verification can provide high-performing stimulus sets in a shorter time frame. This statement is confirmed by the results in Table 2, where both sets of random generations achieved 100% coverage fulfillment significantly more quickly than the implemented genetic algorithm-based approaches. However, if the coverage target is not easy to fulfil by a randomized approach (i.e., the score of constrained-random generation (CRG) of stimuli used for coverage fulfillment in case of ALU DUT did not exceed 86.7%), genetic algorithms represent the ideal solution (see Tables 3 and 4). For both easily reachable and more difficult to reach coverage targets, the importance of random generation cannot be neglected. The performance of genetic algorithms is greatly increased if the initial population is well-randomized.

Additionally, in complex DUTs there are a high variety of coverage targets. Those which are simpler to achieve can be approached using classical CRG; however, the more complicated coverage targets involve the development of more advanced methods for generating stimuli, such as GA. Hence, we consider that the combination of CRG and GA approaches can be the best recipe for achieving coverage fulfillment for a DUT with the lowest effort.

The current work offers several types of GA implementations, which were tested in three different examples, conferring the advantage of analyzing the problem of coverage fulfillment using additional points of view. For this reason, the obtained results can help in obtaining better results for a wider range of verification targets.

Mutation represents a genetic operator that can help genetic algorithm approaches to avoid local maxima and search for the global maximum. Its application proves beneficial depending on how often mutation is performed in different contexts. For example, in the case of v3.3, the change in the percentage of mutation rate from 2% to (1 divided by the number of generations %) lowered the performance obtained by v3, as shown in Table 3.

Another interesting result is notable by analyzing the outcome of the v3.1 approach in Table 4, (where the lowest performance, only 80% coverage fulfillment, was found, and the v3.5 approach in Table 6, where 20 coverage intervals were used and where the lowest performance, 85% coverage fulfillment, was found. Combining parents randomly and not combining the best parents with the best parents, medium parents with medium parents and low-performance parents with low-performance parents provided bad results. A similar conclusion was proven by the development and testing of v3.4 (Table 6), where each coverage score of a high-performing parent was combined with the score of an opposite individual. Due to of random generation of individuals (here it must again be noted that an initial population of well-distributed values is able to highly boost the performance of a GA), exceptions to this rule can be expected to appear, as seen in the performance of the v3.5 approach, when 15 coverage intervals were employed and the value of 100% coverage was obtained after only five generations (Table 6).

As expected, if each generation embeds more individuals, better results are obtained (see v3.3 performance for 20 coverage intervals in Table 6). However, the time needed for running the program evolves in an exponential manner in this case. Although a generation with 100 individuals represented a successful combination of constrained random verification and genetic algorithms, in this work, generations of 20 individuals were used. This approach allowed us to better compare the performance of different genetic algorithm-based approaches, diminishing the contribution of randomness in the generation of individuals. However, a mutation that alters an item at a random position remained very important for coverage fulfillment automation performance in the current work.

There are multiple ways to configure the evolution of populations, and this paper provides only a few examples; other approaches are worth attempting. As demonstrated in this study, the same approach can perform better in some cases (v3.2 outperformed v3.3 when the ALU design was the focus) while having lower performance in other situations

(version 3.3 exceeded the performance of v3.2 in the case of the next two presented projects). However, v3.2 and v3.3 can both be considered the best in the current study. These versions managed to achieve high values of coverage for all three tested DUTs. In all cases, these approaches obtained better performance than random simulations depicting constrained-random verification.

6. Conclusions

Using genetic algorithms, we showed how DUT functional coverage can be fulfilled by offering a set of approaches and observing the results. The work presented here would not have been possible without creating a mechanism for corelating the input stimulus sets with their obtained coverage score. One advantage of using genetic algorithms is that even with a high number of stimuli, a close to maximum coverage value can be achieved in reasonable time.

By using the presented GA-based approaches, multiple sequences of stimuli able to reach the desired coverage value were discovered and stored for further use. A potential industrial application of the proposed method is to replace the ranked regressions generated by commercial simulators with verification tests which run the stimulus sequences found here. Ranked regressions contain all of the tests which can be used to obtain a high coverage value. If a test or several tests developed using a genetic algorithm approach can replace ranked regression, a significant amount of simulation time can be saved. By generating shorter tests using only the input data contained by an individual and thereby achieving a high coverage value, another aim of the current work was successfully met. In addition, by transferring to the algorithms a part of the tedious work which in most cases is currently accomplished by verification engineers (when directed tests are created for fulfilling the last missing fractions of coverage percentage), the verification job itself becomes more pleasant for the engineers.

Overall, given the scenarios analyzed in the current paper, the GA based approaches outperformed CRG in achieving the maximum coverage value for the analyzed targets. CRG is the first step in the industry-standard mechanism used for generation of stimuli in functional verification, with the second step being the creation of directed tests, which is supported by UVM methodology as well. In Table 2, as well as in all tables summarizing the results, CRG did not manage to deliver 100% coverage fulfillment.

However, the above-mentioned benefits of GA do not wholly take CRG out of the landscape of coverage fulfillment tasks. CRG is considered the best candidate for easy-to-achieve coverage objectives, as it requires the least effort when using a tool that provides good randomization mechanisms. Moreover, application of mutation (an important representative of randomness within genetic algorithm-based approaches) to alter an item at an individual's random position retains a steady place in coverage fulfillment automation powered by GA.

Considering the work presented in this paper, genetic algorithms can be seen as a technique with high potential for increasing levels of coverage fulfillment. However, because they require several iterations to reach maximum coverage value, they may not fit all verification needs. Testing the performance of approaches based on genetic algorithms in more complex scenarios is required in order to control multiple streams of DUT stimuli. This topic represents a further step to be accomplished in the journey toward automation of functional verification using artificial intelligence techniques.

Author Contributions: Conceptualization, G.M.D. and A.D.; methodology, G.M.D. and A.D.; software, G.M.D. and A.D.; validation, A.D.; formal analysis, G.M.D. and A.D.; investigation, A.D.; resources, G.M.D. and A.D.; data curation, A.D.; writing—original draft preparation, A.D.; writing—review and editing, G.M.D. and A.D.; visualization, G.M.D. and A.D.; supervision, A.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dinu, A.; Craciun, A.; Alexandru, M. Hardware reconfiguration of a SoC. *Rev. Air Force Acad.* **2018**, *1*, 55–64. [CrossRef]
2. Yuan, F.L.; Wang, C.C.; Yu, T.H.; Marković, D. A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing. *IEEE J. Solid-State Circuits* **2014**, *50*, 137–149. [CrossRef]
3. Dinu, A.; Danciu, G.M.; Ogruțan, P.L. Debug FPGA projects using machine learning. In Proceedings of the 2020 International Semiconductor Conference (CAS), Sinaia, Romania, 7–9 October 2020; pp. 173–176.
4. Dinu, A.; Ogrutan, P.L. Opportunities of using artificial intelligence in hardware verification. In Proceedings of the IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME), Cluj-Napoca, Romania, 23–26 October 2019; pp. 224–227.
5. Piziali, A. *Functional Verification Coverage Measurement and Analysis*; Springer Science & Business Media: New York, NY, USA, 2007.
6. Anilkumar, R.; Varaprasad, B.K.S.V.L.; Padmapriya, K. Automation of Translating Unit-Level Verification Scenarios for Test Vector Generation of SoC. In *Intelligent Sustainable Systems*; Springer: Singapore, 2022; pp. 527–536.
7. Herdt, V.; Drechsler, R. Advanced virtual prototyping for cyber-physical systems using RISC-V: Implementation, verification and challenges. *Sci. China Inf. Sci.* **2022**, *65*, 1–7. [CrossRef]
8. Krylov, G.; Friedman, E.G. EDA for Superconductive Electronics. In *Single Flux Quantum Integrated Circuit Design*; Springer: Cham, Switzerland, 2022; pp. 95–114.
9. Foster, H.; Krolnik, A.; Lacey, D. Functional Coverage. In *Assertion-Based Design*; Springer: Boston, MA, USA, 2003; pp. 123–159.
10. Dinu, A.; Ogruțan, P.L. Coverage fulfillment methods as key points in functional verification of integrated circuits. In Proceedings of the 42th International Semiconductor Conference (CAS), Sinaia, Romania, 9–11 October 2019; pp. 199–202.
11. Ismail, K.A.; Abd El Ghany, M.A. High Performance Machine Learning Models for Functional Verification of Hardware Designs. In Proceedings of the 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 23–25 October 2021; pp. 15–18.
12. Ismail, K.A.; Ghany, M.A. Survey on Machine Learning Algorithms Enhancing the Functional Verification Process. *Electronics* **2021**, *10*, 2688. [CrossRef]
13. Ioannides, C.; Eder, K.I. Coverage-directed test generation automated by machine learning—A review. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2012**, *17*, 1–21. [CrossRef]
14. Cristescu, M.-C. Machine Learning Techniques for Improving the Performance Metrics of Functional Verification. *Sci. Technol.* **2021**, *24*, 99–116.
15. Samarah, A.; Habibi, A.; Tahar, S.; Kharma, N. Automated Coverage Directed Test Generation Using a Cell-Based Genetic Algorithm. In Proceedings of the IEEE International High Level Design Validation and Test Workshop, Monterey, CA, USA, 8–10 November 2006; pp. 19–26.
16. Van Rossum, G. Python Programming Language. In Proceedings of the USENIX Annual Technical Conference, Santa Clara, CA, USA, 17–22 June 2007; Volume 41, p. 36.
17. Lutz, M. *Programming Python*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2001.
18. Ferreira, A.; Franco, R.; da Silva, K.R.G. Using Genetic Algorithm in Functional Verification to Reach High Level Functional Coverage. Available online: https://www.inf.ufrgs.br/sim-emicro/papers/sim2013_submission_50.pdf (accessed on 14 December 2021).
19. Franco, R.A.P.; da Silva, K.R.G.; Rodrigues, C.L. Genetic Algorithm applied to the Functional Verification in Digital Systems. *J. Integr. Circuits Syst.* **2018**, *13*, 1–9. [CrossRef]
20. Cristescu, M.-C.; Bob, C. Flexible Framework for Stimuli Redundancy Reduction in Functional Verification Using Artificial Neural Networks. In Proceedings of the IEEE International Symposium on Signals, Circuits and Systems (ISSCS), Daegu, Korea, 22–28 May 2021; pp. 1–4.
21. Subedha, V.; Sridhar, S. An efficient coverage driven functional verification system based on genetic algorithm. *Eur. J. Sci. Res.* **2012**, *81*, 533–542.
22. Cihan, A.N.; Güğül, G.N. An Indoor Smart Lamp For Environments Illuminated Day Time. In Proceedings of the IEEE East-West Design & Test Symposium (EWDTS), Varna, Bulgaria, 4–7 September 2020; pp. 1–5.
23. AlFaris, F.; Juaidi, A.; Manzano-Agugliaro, F. Intelligent homes' technologies to optimize the energy performance for the net zero energy home. *Energy Build.* **2017**, *153*, 262–274. [CrossRef]
24. Accelera. Universal Verification Methodology (UVM) 1.2 User's Guide. Available online: https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf (accessed on 14 December 2021).
25. Bergeron, J. *Writing Testbenches Using System Verilog*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2007.
26. Modelsim. Available online: https://eda.sw.siemens.com/en-US/ic/modelsim/ (accessed on 14 December 2021).
27. Stefan, G.; Alexandru, D. Controlling hardware design behavior using Python based machine learning algorithms. In Proceedings of the 16th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 10–11 June 2021; pp. 1–4.

28. Ji, X.; Ye, H.; Zhou, J.; Yin, Y.; Shen, X. An improved teaching-learning-based optimization algorithm and its application to a combinatorial optimization problem in foundry industry. *Appl. Soft Comput.* **2017**, *57*, 504–516. [CrossRef]
29. Nugraheni, C.E.; Abednego, L.; Widyarini, M. A Combination of Palmer Algorithm and Gupta Algorithm for Scheduling Problem in Apparel Industry. *Int. J. Fuzzy Log. Syst.* **2021**, *11*, 19–33. [CrossRef]
30. Eiben, A.E.; Raue, P.-E.; Ruttkay, Z. Genetic algorithms with multi-parent recombination. In Proceedings of the International Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, 9–14 October 1994; Springer: Berlin/Heidelberg, Germany; pp. 78–87.