INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Artificial Intelligence and Embedded Systems Laboratories

Generating Optimal Functional Coverages in Digital Systems

THESIS

A doctoral dissertation submitted by

M. Sc. Alfonso Martínez Cruz

For the degree of

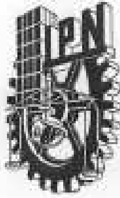
Doctor of Computer Science

Advisors

Ricardo Barrón Fernández, Ph.D.

Herón Molina Lozano, Ph.D.

Mexico city, January 2016



SIP-14 BIS

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 14:00 horas del día 4 del mes de diciembre de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Generating Optimal Functional Coverages in Digital Systems"

Presentada por el alumno:

MARTÍNEZ	**CRUZ**	**ALFONSO**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: | A | 1 | 2 | 0 | 9 | 0 | 2 |

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis

Dr. Ricardo Barrón Fernández

Dr. Herón Molina Lozano

Dr. Sergio Suárez Guerra

Dr. Víctor Hugo Ponce Ponce

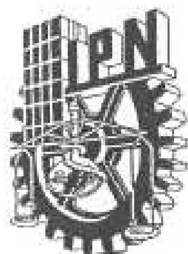
Dr. Salvador Godoy Calderón

Dr. Marco Antonio Ramírez Salinas

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
DIRECCIÓN



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, el día 7 del mes Enero del año 2016, el que suscribe Alfonso Martínez Cruz alumno del Programa de Doctorado en Ciencias de la computación con número de registro A120902, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Ricardo Barrón Fernández y Dr. Herón Molina Lozano, y cede los derechos del trabajo intitulado Generating Optimal Functional Coverages in Digital Systems, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección mcruzb09@sagitario.cic.ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Alfonso Martinez Cruz

Resumen

El constante avance de la tecnología y los nuevos requerimientos en el desempeño y producción de los sistemas digitales han requerido que nuevos esquemas de verificación y prueba sean propuestos. Como hace unas décadas Gordon E. Moore predijo que el número de componentes transistores en un circuito integrado se duplicaría cada dos años así como la tendencia del desarrollo de nuevos dispositivos cada ves mas pequeños y con mayor complejidad en su funcionamiento han producido que hoy en día, tengamos a nuestro alcance dispositivos como: smart-phones, smart-watches, tabletas, drones, computadoras, etc. que contienen chips con millones de dispositivos semiconductores procesando gran cantidad de información y realizando diferentes funciones en pequeños intervalos de tiempo.

La verificación funcional representa una parte importante en el proceso de diseño de los sistemas digitales debido a que los errores en hardware resultan más costosos y es necesario remplazar los dispositivos físicamente. En el presente trabajo de investigación se propone un nuevo método de generación de altos porcentajes de coberturas funcionales para la verificación de sistemas digitales. El método propuesto esta basado en la aplicación de meta-heurísticas (Particle Swarm Optimización, Differential evolution algorithm), el uso de modelos de cobertura funcional y la simulación de los dispositivos bajo verificación. Dicho método representa un método hibrido y a la ves una alternativa para complementar las técnicas actuales utilizadas para realizar la verificación funcional.

A diferencia de los trabajos previos donde se han utilizado algoritmos genéticos, en esta investigación otras meta-heurísticas como: Particle Swarm Optimización algorithm, Differential Evolution algorithm son utilizadas para la generación de vectores de prueba que maximicen los valores de cobertura funcional. Estas meta-heurísticas han sido utilizadas en diferentes problemas de optimización obteniendo resultados competitivos. También, en esta investigación se propone una nueva versión del algoritmo de evolución diferencial compacto para la representación binaria. Dicho algoritmo esta basado en el principio del algoritmo genético compacto ya que utiliza los valores estadísticos para representar a la población, por lo tanto, requiere menos recursos de memoria para su implementación que los algoritmos basados en la población.

Para realizar la prueba de la implementación a nivel RTL han sido diseñadas diferentes herramientas de software. Debido a esto, una plataforma de software ha sido diseñada para

conectar las herramientas de simulación con los módulos en lenguaje de alto nivel. Esta plataforma esta basada en los esquemas actuales para llevar a cabo la verificación por medio de de la simulación de los dispositivos. La plataforma contiene un conjunto de módulos que permiten utilizar diferentes algoritmos, escenarios de prueba, asi como conectar las implementaciones de los dispositivos con el entorno de pruebas.

Abstract

The constant advance of technology and new requirements in the performance and production of new digital systems have required new testing, and verification schemes. A few decades ago Gordon E. Moore predicted that the number of transistors on the components of an integrated circuit would double every two years. Moreover, the new development trends in devices allow for even smaller circuits, that perform more complex operations, permitting the existence of devices such as smart-phones, smart-watches, tablets, drones, computers, etc. containing chips with millions of semiconductors, with a wealth of information processing and performing many different functions in short time periods.

Functional verification represents an important design process partly because the errors in hardware are very expensive and necessitate the replacement of the device. In this research a new test generation method is proposed in order to obtain high rates of coverage for functional verification of digital systems. The proposed method is based on the application of meta-heuristic algorithms (Particle Swarm Optimization, Differential evolution algorithm), using functional coverage models and simulation devices under verification. This is a hybrid method and represents an alternative complementing existing techniques used for functional verification.

Unlike previous studies where genetic algorithms have been used, this research provides other meta-heuristics such as a Particle Swarm Optimization algorithm, and a differential evolution algorithm which are used to generate test vectors that maximize the functional coverage values. These meta-heuristics have been used in different optimization problems obtaining competitive results. Moreover, this research proposes a new version of the compact differential evolution algorithm for binary representation. This algorithm is based on the principle of a compact genetic algorithm that uses statistical information in order to represent the population; therefore, it requires less memory resources for implementation than algorithms based on population.

In order to perform the device implementation test in a Register Transfer Level (RTL) different software tools have been designed. Due to this, a software platform is proposed to connect the simulation tools with the modules in high level language. This platform is based on current schemes to conduct verification through simulation devices. The platform contains a set of modules that may use different algorithms, and test scenarios, as well as implementations connecting devices to the test environment.

Agradecimientos

GRACIAS *A mis padres Josefina y Gabriel y a mi familia por darme la oportunidad de crecer, desarrollarme y cumplir mis metas.*

GRACIAS *A mis directores de tesis: Dr. Ricardo y Dr. Heron por guiarme y compartir sus experiencias durante mis estudios de doctorado.*

GRACIAS *Al profesor Kwang-Ting (Tim) Cheng por todo su apoyo y sus enseñanzas durante mi estancia en la Universidad de California en Santa Barbara (UCSB).*

GRACIAS *Al Instituto Politécnico Nacional por permitirme realizar mi formación durante mis estudios de doctorado.*

GRACIAS *A mis profesores durante mi estancia en el Centro de Investigación en Computación por compartir su conocimiento y formar parte de mi preparación en mis estudios de doctorado.*

GRACIAS *Al Consejo Nacional de Ciencia y Tecnología, CONACyT, por el apoyo económico proporcionado durante la realización de mis estudios de doctorado. Agradezco a la Secretaria de Investigación y Posgrado del Instituto Politécnico Nacional a través de los proyectos SIP-20151625 y SIP-20151454.*

GRACIAS *A mis compañeros y amigos por compartir sus consejos, experiencias y por su apoyo durante mis estudios. A Lawrence y Bonnie Blakley por su apoyo y atencion durante este trabajo.*

"Humility, preparation and your actions make you a better person."
Alfonso Martínez Cruz

"Sometimes when we want to be better, we need to keep in a box the goals we have already achieved. That box is always carried with us, because we will have difficulties on the road and at that time, the goals in that box will be links to achieve a new proposed goal."

alfonso martínez

Content

Resumen I

Abstract III

Agradecimientos V

Figures XII

Tables XIII

I Introduction 1

1 Introduction 3

1.1 Functional Verification of Digital Systems 5
1.1.1 Elements of Functional Verification 8
1.2 Problem to Solve 9
1.3 Justification 11
1.4 General Objective 11
1.4.1 Specific Objectives 11
1.5 Research and developed method 11
1.6 Scope of work 12
1.7 Contributions 12
1.8 Organization 13
1.9 Resume 13

II State of the Art 15

2 State of the Art 17
2.1 Introduction . . . 17
2.2 Coverage Directed Test Generation for Functional Verification . . . 18
2.3 Meta-heuristics algorithms . . . 20
2.4 Bayesian Networks and Markov Model approaches . . . 22
2.5 Methods based on mutations . . . 24
2.6 Methods based on data mining . . . 25
2.7 Functional Test Generation based on extraction of State Machines . . . 27
2.8 Methods based on Theorem Proving . . . 28
2.9 Methods based on Model Checking . . . 29
2.10 Resume . . . 31

III Digital Design and Functional Verification of Digital Systems 33

3 Digital Design and Functional Verification of Digital Systems 35
3.1 Digital systems design . . . 36
3.2 Main elements of Functional Verification . . . 39
3.2.1 Functional Coverage . . . 41
3.2.2 Coverage Points . . . 43
3.2.3 Functional Coverage Metrics . . . 45
3.2.4 Calculation of Functional Coverage . . . 47
3.2.5 Functional Coverage Model description . . . 49
3.2.6 Directed Functional Verification for Digital Systems . . . 51
3.2.7 Controllability and Observability in digital systems . . . 54
3.2.8 Pseudo-random Constraint Stimulus . . . 55
3.2.9 Modeling Functional Verification through HDL . . . 57
3.2.10 Proving the Design through the Simulation . . . 57
3.3 Resume . . . 59

IV Meta-Heuristics on the Binary Space 61

4 Meta-Heuristics on the Binary Space 63
4.1 Binary search space 64
4.2 Evolutionary algorithms 65
4.2.1 Binary Genetic Algorithm 66
4.2.2 Binary Particle Swarm Optimization algorithm (BinPSO) 68
4.2.3 Binary Differential Evolution Algorithm (DE Algorithm) 70
4.3 Compact meta-heuristic algorithms 72
4.3.1 Compact Binary Genetic algorithm 74
4.4 Resume 76

V Proposed Method 77

5 Proposed Method 79
5.1 Proposed Compact Binary Differential Evolution Algorithm 80
5.2 Proposed test vector generation method 84
5.2.1 Fitness functions 87
5.2.2 Verification platform 88
5.2.3 Applying meta-heuristic algorithms over the proposed platform 91
5.2.4 Modeling the Device Under Verification 93
5.2.5 Calculating and analyzing Functional Coverage 95
5.2.6 HDL Simulation of the Device Under Verification 97
5.3 Resume 98

VI Experiments and Results 101

6 Experiments and Results 103
6.1 Case Study 104
6.1.1 Experimental Setup 105
6.1.2 Experiments and Results with Compact-BinDE Algorithm 106
6.2 Experiments and Results with other Meta-heuristic Algorithms 115
6.2.1 Experiments using a binary Genetic algorithm 116

6.2.2 Experiments using Binary Particle Swarm Optimization algorithm . . . 119
6.2.3 Comparison between algorithms 122
6.3 Discussion 126

Conclusions and future work **129**

7 Conclusions and future work **131**
7.0.1 Conclusions 132
7.0.2 Future Work 135
7.1 Contributions 136
7.1.1 Published papers in Journals 136
7.1.2 Published papers in conferences 136

Glossary **139**

Bibliography **148**

Figures

1.1 Testing vs Verification. 6
1.2 Functional Verification based on simulation vs Formal Verification. 7
1.3 General methodology for the functional verification process. 9

2.1 Coverage Directed Test Generation schema. 19

3.1 Design Intention diagram. 36
3.2 Main steps of digital system design. 40
3.3 General methodology for the functional verification process. 41
3.4 Trends in Functional Verification Techniques. 43
3.5 Model representation of a DUV (UART-bus) using the coverage points. 50
3.6 Coverage Directed Test Generation blocks diagram. 52
3.7 Observability and Controllability in a Digital system. 55
3.8 Constraint random test generation. 56
3.9 Modeling a device through HDL simulation. 58
3.10 Flowchart of verification based on simulation. 59
3.11 Trends in Functional Verification Techniques. 60

4.1 Generation of new vectors in the Binary Differential Evolution algorithm. . . . 72

5.1 Blocks diagram of the test sequences generation method using CoverPoints. . . 86
5.2 Verification interface using the proposed Compact-BinDE algorithm. 90
5.3 Schema of the designed platform in order to perform functional verification. . . 91
5.4 Flow Diagram binary GA algorithm using the verification platform. 92
5.5 Flow Diagram PSO algorithm used for generating sequences of vectors. 93
5.6 Flow Diagram Compact-BinDE for generating sequences of vectors. 94
5.7 Model representation of a DUV (UART-bus) using coverage points 95
5.8 Manual process to perform the Directed Coverage Functional Verification. . . . 96
5.9 Automated process to perform the Functional Verification. 96
5.10 Implemented process to perform the Functional Verification. 99
5.11 Modelsim simulator used in the experiments. 100

6.1 Coverage values obtained using f_1 and the DE/rand/1/bin. 112
6.2 Comparison of Coverage Directed Test Generation for different scenarios. 113
6.3 Coverage Directed Test Generation percentages for different iterations. 114
6.4 Comparison of Coverage Directed Test Generation with different CR values. . . . 115
6.5 Comparison of Coverage Directed Test Generation using $f1$ and $f2$. 116

7.1 Coverage space example. 141

Tables

2.1 Works using meta-heuristic algorithms to perform functional verification 22

6.1 Parameters of four different scenarios using Compact-BinDE algorithm 107
6.2 Coverage values obtained with four different configurations 107
6.3 Parameters of four best scenarios using the Compact-BinDE algorithm. 108
6.4 Values of Coverage percentage obtained with different population size 108
6.5 Parameters of four best scenarios using a solution length of two sequences . . . 109
6.6 Coverage and total time for using different mutation functions 109
6.7 Parameters of four best scenarios using the UART-IP device 110
6.8 Obtained results for four scenarios using Compact-BinDE algorithm 110
6.9 Compact-BinDE parameters using a length of two sequences for UART-IP . . . 111
6.10 Results using different configurations of the Compact-BinDE algorithm 112
6.11 Parameters for experiment using binary Genetic algorithm for FIFO memory . 117
6.12 Results obtained using the binary Genetic algorithm for FIFO memory 117
6.13 Parameters for experiment using binary Genetic algorithm 118
6.14 Results obtained using the binary Genetic algorithm 119
6.15 Algorithm parameter values used for PSO scenarios with FIFO memory 120
6.16 Results obtained for experiment 1 using a FIFO memory 121
6.17 Parameters for Particle Swarm Optimization algorithm using a UART device . . 121
6.18 Results obtained using the PSO algorithm with a UART bus core 122
6.19 Results obtained using SR, GA, Compact-BinDE and PSO algorithm 123
6.20 Parameters for experiment using binary Genetic algorithm 124
6.21 Results obtained using GA, SR and the Compact-BinDE algorithm 124
6.22 Results obtained using SR, GA, Compact-BinDE and PSO algorithm 126

CHAPTER I

Introduction

Chapter 1

Introduction

Current digital systems have a large amount of resources, performing millions of operations per second. Circuits included in smart-phones, laptops, tablets, etc. permit complex devices to have greater functionality complexity. Due to this, the digital design has a compromise with the market times and technological advances. During the digital systems design process, several steps are involved; one of the most important is design verification. The verification represents the biggest part of the total devices manufacturing cost. The approximate cost of this phase is estimated at about 60% of the total digital system design cost. Unlike software, the errors in hardware designs are very expensive, because these require redesigning and physically replacing the failed device. Also, because of the increase in the functionality complexity and number of transistors of digital systems lately, the importance of creating efficient designs and reducing their verification time has generated the need to create more efficient design verification techniques. It means, techniques that reduce the time and increase the test coverage percentage and because of this, different methods have been developed to perform the verification processes and new software tools have been produced. Furthermore, these software platforms have been used with different digital systems, and the results obtained have shown that it is necessary to develop more efficient methods that can cover all design cases.

The task of checking all input vectors in the device is unfeasible because the coverage space increases as the device complexity also grows. Generally, the main objective of the informal verification techniques is to increase design space coverage and changes for finding errors in the digital systems design. When the verification of an implemented device is performed in a hardware description language (Verilog, SystemC, SystemVerilog, etc), the

verification engineers need to use specialized software tools (Computer Aided Tools-CAD, Questa, Modelsim, Cadence, etc).

Lately, the complexity of digital systems has resulted in the need for new techniques and tools. Different Electronic Design Automation (EDA) Companies provide different software tools in order to perform the steps involved in the functional verification. Different conferences like Digital Automation Conference (DAC), Design Automation and Test in Europe (DATE) have been created in order to publish and share the recent advances concerning digital design, automation techniques among other topics. Researches from industry and academics have published different works related to the verification topic.

Directed Functional Verification has an important role to meet with the conditions during functional verification. It has been detected that the pseudo-random test generation methods are not effective to cover hard cases, therefore, it is necessary to propose new test vector generation methods which can make an efficient search and exercise all functionality.

Three different philosophies have been proposed in order to perform the Functional Verification: Static methods (formal methods), Dynamic methods (Which are based on simulation) and Hybrid methods (which do not fall in formal and informal methods).

Every philosophy contains different strategies in order to test the functionality of a digital system. However, at present, none of these methodologies have been enough to over pass the different problems because the complexity of the functionality of digital systems is increasing. New challenges have appeared, some of them are: compatibility of modules, synchronization of clocks, among others.

When functional verification is performed, one or more coverage models are used. These coverage models are based on a type of coverage structure or coverage metrics, i.e., finite state machine (FSM), statement, branch path, expression coverage, etc. The coverage model is a fundamental piece for the verification methods representing a golden model to describe the device behavior. Different ways to close the loop between the end of simulation and the new test generation have been proposed. Coverage Directed Test Generation (CDG) has been proposed as a possible solution to this problem. Different experiments have shown that directed probes are promising because a small number of them can reach the same coverage goal with respect to the constrained random probes.

In this research we propose a new method which uses reduced binary meta-heuristics in order to generate sets of test vector sequences. We focus on the hybrid methods (based on the simulation and meta-heuristics) since these methods have obtained good results even though

there is an increase in digital systems complexity. The strategy employed is based on the use of coverage models for the devices verification process, which are built with relevant conditions or coverage points representing the Device Under Verification (DUV) full behavior. The main problem consists in covering all hard cases since the relationships between the CoverPoints and the input data at the design are not trivial. Different to previous works that used meta-heuristics, the proposed method can reduce the number of evaluations used to obtain test sequences that exercise the coverage points.

This chapter presents an introduction to the Functional Verification topic, the problem to solve, the objectives, the justification and the contributions of the present work.

1.1 Functional Verification of Digital Systems

Functional Verification can be described as the application of information theory where a redundancy and error correction code are required in order to keep the integrity of the information through the design cycle.

Different from the traditional applications of information theory where the message is preserved as it is sent through the communication channel, it is intentionally refined and becomes less abstract during every transformation through the design process. The design process can be described as: incremental clarification put into the communication channel in every stage of the design [1].

In most cases the definition of the Testing and Functional Verification is confused. The testing can be defined as a set of tests applied to the Device Under Verification (DUV) in order to determine if the behavior in each test meets the specification. It is a sample process where not all aspects of the device are exercised, which means that a total subset of all possible device characteristics is used.

In other words, testing consists of a set of stimulus applied to the device in order to test a particular test case or analyze the response of the performance based on the expected behavior. Figure 1.1 shows the testing and the verification into the design process of Digital Systems.

Functional Verification is a comparative process. It includes a wide set of techniques in order to find faults in the behavior of a device. Functional Verification shows if the hardware or software meets the original specification requirements. Also, the functional verification does not show the fault itself. It merely shows the presence of an error.

With the implementation of pseudo-random verification around 1980, it was possible to

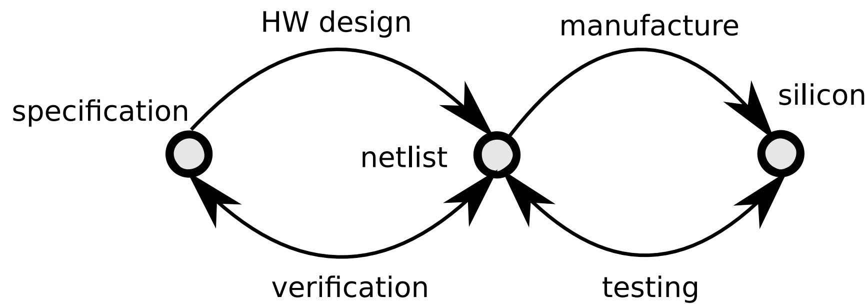


Figure 1.1. Testing vs Verification.

explore the design limits using an automated test in order to exercise a range of inherent variability in the functionality of a Digital System. The standard verification consists of:

- A standard context to the analysis which is used for the principles of the pseudo-random functional verification.
- Standard inherent variables in the design.
- Standard interpretation of the specification which is used to define the variables and ranges, rules and guidelines.

Simulation is most frequently used in Functional Verification of Digital Systems because it is not necessary to build a complex model, or use sophisticated techniques. Figure 1.2 shows the verification which is performed by means of simulation and formal verification of a digital system. The dynamic verification (based on simulation) uses a coverage model which represents a golden model because this contains all device functionality. This model is compared with the results obtained at the end of the simulation which are saved in the regression suit (where suit is a database). In every iteration a set of directed tests is produced by a test generator module using the information from the regression suit.

An important aspect of the Functional Verification is the functional coverage which can be defined as the percentage of verification objectives that have to be met. This is used as a metric for evaluating the progress of a verification project in order to reduce the number of simulation cycles expected in verifying a design. The coverage should be restricted to only those values that could indicate the design is fully verified. Also, the functional coverage is used to verify the correct operation of a device by means of the representation of a coverage model that contains one or more coverage points.

The coverage points are grouped in sets which are best known as coverage groups. A coverage group is a set of attributes, grouped together for purposes of deployment in a common

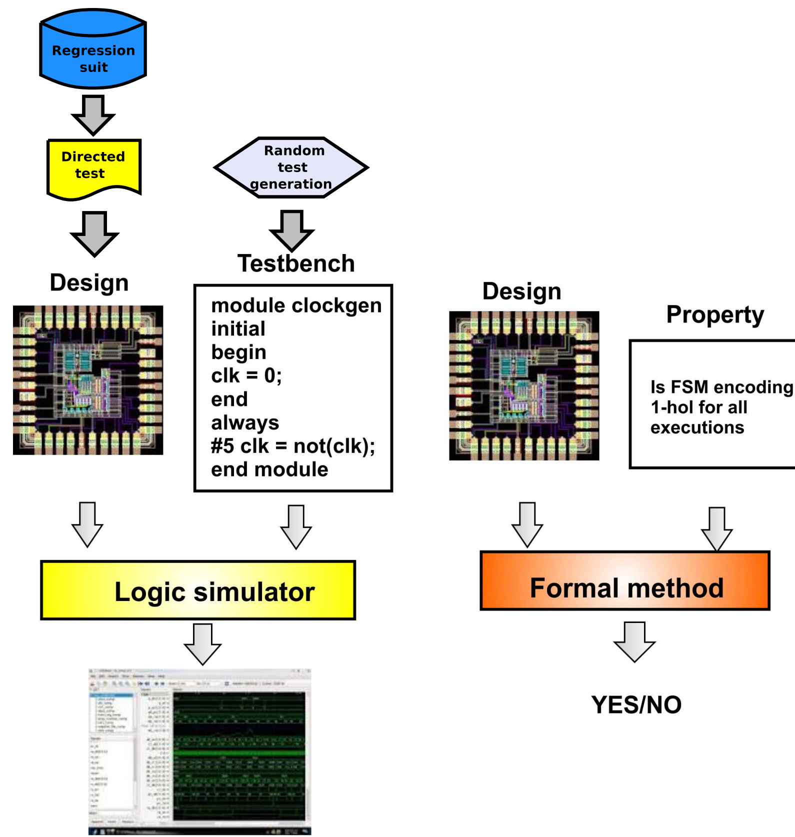


Figure 1.2. Functional Verification based on simulation vs Formal Verification.

correlation time. Also, a coverage group may contain different elements for different functions to verify the specification of a coverage model. These elements may include a clock event, a set of points, and weights for points, among others.

A functional coverage point can be a scalar value, a condition or an expression in a digital system. For example, the "output data" variable, since it is a variable that can take a set of values $(0, 1, 2, ..2^n - 1)$. These values are better known as *bins* (There is a glossary at the end the thesis). Examples of coverage points include: the level of occupancy of a buffer, an instruction code, address write/read in a memory, an input to a register, package longitude, etc. The objective of a coverage point is to ensure that all interesting and relevant values are observed in the sampled value or expression. Moreover, a coverage hole is a point which has not been exercised or tested during the functional process. After a device simulation it is important to analyze the produced information in order to review the exercised points and the holes that remain.

The coverage means the measure of the integrity of a test set. The coverage definition

represents the number of met goals by the test set which is used. Also, the functional coverage is defined as the number of defined goals which are met during the verification process.

During the verification process the verification engineers use a set of metrics in order to measure the validation process based on the simulation. At the beginning, they use basic metrics which require little effort, after that, they use more sophisticated metrics. A coverage metric can be defined as a parameter or attribute which is used as a unit in order to measure the verification process in one dimension. There are different kinds of metrics: code coverage, jump coverage, trajectory coverage, sentences coverage, etc. In the next subsection some aspects about functional verification are described.

1.1.1 Elements of Functional Verification

When Functional Verification for digital systems is performed, different elements are involved. These elements include: A functional specification, a verification plan, the implementation of the device, among others. A functional specification is the common source for the implementation and the verification of a device. Generally, it is implemented in two different documents with different abstraction levels:

1) The first one describes the requirements of the architecture to be implemented. It details the functions to be met by the device.
2) The second one is the specification of the device implementation. It describes the implementation of the device architect in a blocks level.

A verification plan is what will be verified and how to collect the verification information. It defines what should be verified and how it should be verified. Moreover, it describes the scope of the verification problem and is used as a functional specification for the test environment.

The device implementation is the design in a register transference level *RTL*, which is modeled in some hardware description language *HDL*. It is based on the specification according to the constraints and the prevision produced by the design engineer.

Different from software, where only the code sentences need to be verified, the hardware designs need to meet the specific times in order to perform the required functionally. Due to this, it is necessary to implement the design in a temporal modeling language.

Figure 1.3 shows the flowchart of a general methodology in order to perform the functional device verification. The methodology of the functional coverage starts by the revision of the specifications of the digital system; after that, the implementation is performed. In order

to perform the Functional Verification, a coverage model is proposed. Given the model, a verification plan is necessary. After the simulation is performed, the results are checked and if a coverage percentage is reached, the process finishes or the test sequence is modified and is used once more.

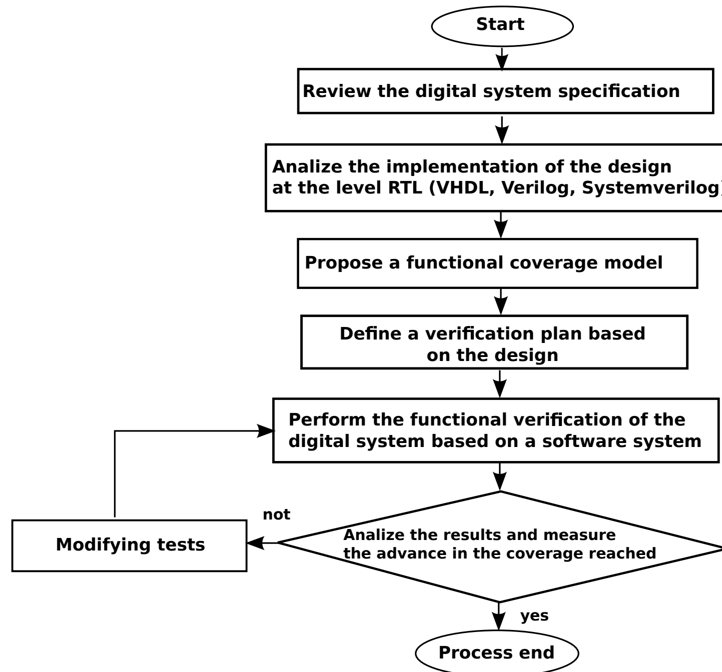


Figure 1.3. General methodology for the functional verification process.

1.2 Problem to Solve

An important area for faults coverage is the Directed Functional Verification. Typically in industry, the digital systems have a set of conditions which should be met. In many environments of industrial designs, the verification engineers are not required to write the formal proprieties in order to prove that the behavior of the system is true. However, test sequences are required to search for errors and exercise the design functionality during the performed process in order to meet all required conditions.

The Directed functional verification performs an important role in order to meet the conditions of the functional specification. Due to the inconsistency of the pseudo-random

test generation methods in order to cover the hard cases, the engineers do not know the requirements of the Directed Functional Verification through simulation, especially with strict market requirements.

Different criteria in order to perform the functional tests generation of a digital system are needed because exhaustive search in a device of small size can require an exorbitant number of test vectors. An example is given in [1] and it is as follows: Given a device which has N inputs and M flip-flops, $(2^N)^M$ vectors can be needed in order to perform the full functional verification. A modest size of a device can have 10 inputs and 100 flip-flops (around 3 registers of 32 bits). This device could require $(2^{10})^{100}$ or 2^{1000} for the full verification. It means if we perform the Functional Verification using 1000 test vectors by second, this can require 339, 540, 588, 380, 062, 907, 492, 466, 172, 668, 391, 072, 376,037, 725, 725, 208, 993, 588, 689, 808, 600, 264, 389, 893, 757, 743, 339, 953, 988, 988, 382, 771, 724, 040, 525, 133, 303, 203, 524, 078, 771, 892, 395, 266, 266, 335, 942, 544, 299, 458, 056, 845, 215, 567, 848, 460, 205, 301, 551, 551, 163, 124, 606, 262, 994, 092, 425, 972, 759, 467, 835, 103, 001, 336, 336, 717, 048, 865, 167, 147, 297, 613, 428, 902, 897, 465, 679, 093, 821, 821, 978, 784, 398, 755, 534, 655, 038, 141, 450, 059, 156, 501 years for the exhaustive test. The functional requirements should be exhaustively tested through formal methods.

In this research the problem to solve consists in finding test binary vector sequences which maximize the functional coverage percentage in a digital system of regular size. It can be defined as follows:

Given a binary space $X = \{0, 1\}$ and a set of vectors $T = \{\vec{t_1}, \vec{t_2}, ..., \vec{t_n}\}$ in order to verify a given digital system with a functional coverage B. The problem consists in finding a set of binary test sequences T' which maximizes B.

Where: $\vec{t_i} \subseteq X, X \equiv \{0, 1\}^n$

Due to the increasing complexity of digital systems, the outperformance of the functional test generation based on the requirements (the functional specification) to be met is needed. Therefore, the problem to solve consists of proposing a method which produces the Directed Test Generation of test vector sequences in order to maximize the obtained coverage and reduce the time which is used to verify a digital system.

1.3 Justification

Currently, the cost of designing and manufacturing process of digital systems is greatly impacted by the step of verification and testing of such systems. Today, it is estimated that the cost of functional verification process involves between 60% and 70% of the total cost. Also, to perform the functional verification of digital systems is an open topic, since due to the high complexity of these systems, time and understanding of the temporary results of the progress of verification are required. That is, why does it take more efficient ways to perform the functional verification process?, and also, how to obtain high values of the total coverage percentage that is verified.

1.4 General Objective

To propose a new test vector generation method in order to generate optimal functional coverages in the Functional Verification of Digital Systems using hardware simulation and meta-heuristic algorithms.

1.4.1 Specific Objectives

- 1.-To design and build a software platform in order to perform the functional verification of Digital Systems.
- 2.-To design a module in order to configure the different strategies of the test vectors generation.
- 3.- To design a new binary compact meta-heuristic algorithm in order to maximize the functional coverage.
- 4.- To compare meta-heuristic algorithms (GA, PSO, Compact-BinDE) and analyze the obtained results.
- 5.- To propose and solve the different optimization problems involved in the generation of optimal coverages for the different digital systems to be verified.

1.5 Research and developed method

The methodology which is used in this research is composed of the following steps:

- To perform the study of the state of the art related.
- To review the specification of digital systems to be tested.
- To analyze the implementation of the design at RTL level (VHDL, Verilog, SystemVerilog).
- To generate a verification plan based on the device to be tested.
- To propose functional coverage models based on the functional specification.
- To propose a heuristic criteria in order to perform the directed functional verification.
- To make the Directed Functional Verification of the digital systems based on a software tool.
- To analyze the obtained results and measure the advance of the functional coverage which is reached.
- To propose modifications of the verification process based on the obtained advances and the digital system which is tested.

1.6 Scope of work

This research develops an automatic verification method in order to perform Directed Functional Verification of digital systems of medium size (ip core bus UART and FIFO memory) that obtain high coverage percentages based on the exercise of most of coverage points from the functional coverage models.

1.7 Contributions

An improved methodology in order to verify the functionality of digital systems based on a hybrid method (meta-heuristic and dynamic methods) which outperforms the original designs based on the verification.

A hybrid method which performs the Directed Test Vector Generation based on meta-heuristic algorithms, coverage models and cost functions.

The application for first time of compact meta-heuristics to the Directed Functional Verification.

A new schema in order to generate optimal functional coverage in digital systems.

1.8 Organization

- **Chapter 1:** The fundamentals of this research are described. These are: Introduction, Problem statement, justification, the main objective and specific objectives, the methodology, the scopes and the contributions.
- **Chapter 2:** Different techniques and methodologies which have been used in order to perform the functional verification of digital systems are given.
- **Chapter 3:** The main definitions about the functional verification area are presented. Moreover, a background is mentioned. Among these definitions are: functional coverage, coverage metric, coverage models, etc.
- **Chapter 4:** The meta-heuristics background on the binary space is highlighted.
- **Chapter 5:** The architecture of the proposed system and the proposed method are described.
- **Chapter 6:** The experiments and obtained results based on the proposed method are shown and analyzed.
- Conclusions and future work are presented according to the obtained results and the development shown in the last chapters.

1.9 Resume

In this chapter the problem to solve, the objectives, the justification and the main problems involved in the scope of this research are presented. On the other hand, a new method of test vector generation based on the union of dynamic techniques (based on simulations of digital systems by means of software tools), and meta-heuristic techniques (Compact binary Differential Evolution algorithm, Particle Swarm Optimization algorithm) is proposed. This method performs the Directed Verification of digital systems. The method uses coverage models which represent the behavior of a digital system by means of a set of coverage points. In order to verify the correct operation of the Device Under Verification (DUV) it is necessary to take into account that one of the main characteristics is the temporal logic because the variables need to be sampled in the correct times. Furthermore, it is necessary to restrict the range of the input values and the amount of tests based on the specification and the device characteristics.

CHAPTER II

State of the Art

Chapter 2

State of the Art

2.1 Introduction

Due to the increase in the complexity of digital systems in recent years, the importance of developing efficient designs and reducing their verification time has generated the need to create more efficient design verification techniques, which reduce the time and increase the test coverage percentage. Because of this, different methods have been developed to perform the verification process. Moreover, new software tools have been produced [2–4]. However, these software platforms were tested with different digital systems, and the obtained results showed that it is necessary to develop more efficient methods that can let us have more confidence and cover the most difficult design cases.

Functional verification has been performed by means of different strategies. Nowadays, there are three different philosophies to perform the functional verification: formal or static methods, dynamic or informal methods and hybrid methods.

Formal, or static methods, use a set of mathematical or logical expressions to represent the device behavior which are based on the sets of formal tests. When these types of methods are applied, a formal software platform is used for verification. These methods include theorem proving, equivalence, and model checking. The advantage of these methods is that all logical functions can be proved based on a formal preposition set [5, 6].

Dynamic methods are based on a run-time hardware simulation using a software platform. Generally, the main objective of these techniques is to increase the coverage of the design space and the logical changes to find mistakes in the design during the simulation. These types of methods are commonly used in industry because they have shown good results in spite of the

increase in the complexity of functionality of digital systems. Furthermore, these methods can test the full functionality, but it is difficult to ensure that the design does not contain errors.

The third category is composed of the hybrid methods which are defined as techniques that do not fall under formal and informal methods because these methods combine both techniques. The main objective of a hybrid method is to address the verification bottleneck by enhancing coverage of the traversed state space combining two or more techniques, but that combination to obtain better results is not trivial.

Another important aspect of the functional verification is the measurement of the progress when the device is being verified. Different coverage metrics are studied during the process in order to measure how well the device functions are. Some works studying the coverage metrics have been proposed [7–10]. Multiple metrics are used because the representation of the coverage models is different from the strategy used.

Due to the fact that the relationships between the inputs and the relevant events or conditions are not trivial when the verification is performed, there are very difficult cases to test during the verification of a digital system. Using all possible binary test input sequences is not a good strategy, especially when the device complexity is increasing, because the simulation time is too long. If a good strategy is not used, then a lot of time is wasted because the same points are tested repeatedly. Therefore, the importance of finding good test sequences to prove all characteristics of device behavior represents one of the biggest challenges to the functional verification. In the next section the introduction about Test Generation for Functional Verification focused on coverage is described.

2.2 Coverage Directed Test Generation for Functional Verification

In order to automate the verification process, a variety of ways to close the loop between the end of simulation and new generation of tests have been proposed. The Coverage Directed Test Generation (CDG) has been proposed as a possible way to automate the verification process. We can define the CDG as a heuristic which searches a finite set of sequences to probe the device behavior using the information obtained at the end of the simulation. Different experiments show that the directed probes are promising because a small number of them can reach the same coverage goal with respect to the constrained random probes. However, when the Coverage

Directed Test Generation is applied, sets of holes are detected at the end of the simulation and one or more conditions need to be tested in a new iteration. This means that one of the main problems consists in exercising all functional coverage events in the device, which is difficult to solve due to the increase of functional complexity in circuits.

Figure 2.1 shows a general schema of Coverage Directed Test Generation for functional verification. In this schema, a set of stimulus scenarios are written in order to test the device. Also, a verification environment is configured. This environment connects the automatic stimulus generation module with the device and the physical layer which is the interface between the device and the generation of stimulus. Finally, when the verification finishes, a coverage report is created with the information about which part of the behavior was exercised. New test cases are written based on the coverage information, this process is repeated until a stop criteria is met.

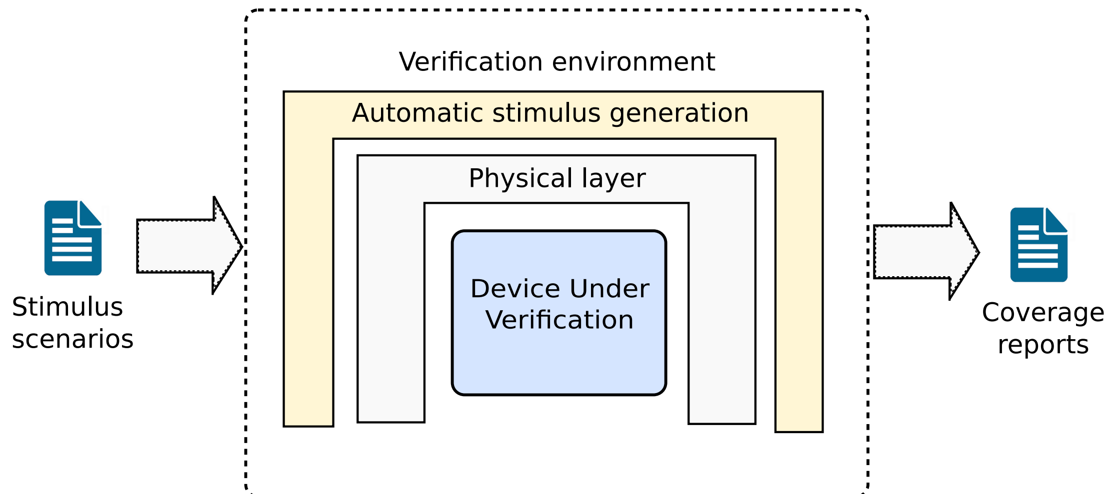


Figure 2.1. Coverage Directed Test Generation schema.

In the next section functional verification works based on meta-heuristic algorithms are presented. In the state-of-the-art that will be presented later different works were based on the use of genetic algorithms and other meta-heuristic algorithms in order to perform the coverage directed test generation schema. However, there are different algorithms that could be used in this area.

2.3 Meta-heuristics algorithms

Searching for sets of test sequences which adequately exercise the functional properties of a system under verification is non-trivial. Deterministic algorithms are exponential in complexity based on circuit size. Meta-heuristic algorithms have been used extensively to manage the growth in complexity in circuit tests. These types of algorithms can be defined as techniques which control one or more heuristics by using mechanisms with a set of parameters. Moreover, these may produce good solutions for problems which cannot be solved in polynomial time in other methods.

In 1975, John Holland published a book (*Adaptation in Natural and Artificial Systems*), which showed that a evolutionary process (genetic algorithm) can be applied to solving different optimization problems. The genetic algorithms use a population of individuals (each one represents a possible solution to a given problem) each is associated to a fitness value. The algorithm transforms the population using the Darwinian principle in each iteration until a good solution is found. The evolutive algorithms have been used in the verification methods to reach higher coverage percentage values.

Some works were published with meta-heuristic methods applied to functional verification, especially using genetic algorithms [11–15]. The genetic algorithm (GA) is a meta-heuristic which has been used to solve different optimization problems in a useful way. These kind of algorithms algorithm manages a population of individuals, using the analogy of the Darwinian theory, where each individual is associated with a fitness value that represents how good is the population to solve a specific problem. After a number of training epochs, the best found solution is an individual which has the best fitness value. In 2001, Mrinal Bose, et al. [13] used a genetic algorithm to perform the verification of a PowerPC architecture. The genetic algorithm biases custom instructions for a pseudo-random generator. The encoding used for chromosome in the genetic algorithm had a fixed length, which represented a sequence of instructions to carry out the system testing. The population size used for the algorithm was small, because the authors concluded that the time cost in the system simulation increases. Better results were found using this technique than using only pseudo-random generation. However, nowadays there are more efficient meta-heuristic optimizer algorithms.

Other work was carried out in 2006, Amer Samarah, et al. [14] used a genetic algorithm to generate directed tests. Each cell was used to represent a chromosome; this means each one represented a random uniform distribution weight over two limits. A set of cells represented

one possible solution. This method was tested with some digital systems using coverage points as a coverage metric, and good results were obtained using this methodology. However, the used representation was very complex and the different configuration values were not automatically generated. Therefore, an extensive knowledge of evolutionary framework by the user is necessary.

Also, in 2008, Haihua Shen, et al. [15] published a paper using a genetic algorithm which was included in a software platform to improve directed functional coverage in a digital system. The chromosome coding form was made based on the instruction set used in the Device Under Verification (DUV). They used a general processor based on an improved Godson1 as DUV. The experiments found better results than the pseudo-random vector generation method. It was concluded that the method helped to achieve non covered tasks and increased the hit rate in hard cover cases.

On the other hand, some methods using the Ant Colony Optimization (ACO) technique have been proposed [16, 17]. The Ant Colony Optimization technique is a meta-heuristic which is based on the swarm intelligence. The algorithm imitates the behavior of the ants finding best paths from the initial state of the food source. The ants search for food by means of the feedback of the pheromone placed by themselves and other ants when they walk through the paths. Also, in 2009, Min Li and Michael S. Hsiao [16] proposed a verification method based on the ACO algorithm. The proposed method used the patches as a coverage metric, and they combined the random generation vectors with a software tool that generates the different states of the digital system. They found that the method could reduce the computational complexity in comparison with the random generation and two heuristics based on the GA algorithms.

Little work has been carried out on techniques for hardware verification based on meta-heuristics. However, now, there are more efficient optimizer meta-heuristic techniques (i.e. Particle Swarm Optimization algorithm (PSO), Differential Evolution algorithm (DE)) [18].

The Differential Evolution algorithm which was proposed by Rainer Storn and Kenneth Price [19]. This is an evolutive algorithm based on the difference between two individuals and a crossover mechanism to generate new possible solutions. One advantage of the algorithm is that only better solutions are used to create a new population. The algorithm has obtained good results in many optimization problems [20]. Moreover, in [21] we propose the use of a binary Differential Evolution algorithm to perform the test sequences generation for the functional verification of a digital system. The main contribution was the application of a binary version describing the behavior through different iteration numbers.

This research differs in that the proposed test generation method performs an efficient binary search by mean of a new compact binary meta-heuristic algorithm (Compact-BinDE) using the principle of Differential Evolution and the compact meta-heuristic algorithms. Also, a Particle Swarm Optimization (PSO) algorithm is used. The proposed strategy uses a hybrid heuristic dynamic verification method to obtain sets of vectors that maximize the functional coverage percentage during the verification of digital systems. The method uses coverage models with sets of points as a functional coverage metric in the binary domain. In the case of Compact-BinDE algorithm, one advantage is that the algorithm performs a reduction of memory requirements, saving only some values of the possible solutions.

Table 2.1 shows some carried out works using meta-heuristic algorithms in order to verify digital systems. In most of works genetic algorithms were used and the obtained results showed that the performance was better than pseudo-random test generation.

Table 2.1. Works using meta-heuristic algorithms to perform functional verification

Year	Author	Title	Device	Coverage
2001	Mrinal Bose	*A Genetic Approach to Automatic Bias Generation for Biased Random Instruction Generation*	load-store reservation station buffer	95 %
2002	Xiaoming Yu	*A Genetic Testing Framework for Digital Integrated Circuits*	Path (code) coverage	67.8 %
2003	Shai Fine	*Coverage Directed Test Generation for Functional Verification using Bayesian Networks*	processor Power PC	94 %
2006	Amer Samarah	*Automated Coverage Directed Test Generation Using a Cell-Based Genetic Algorithm*	Specman e Simple CPU and Router models	100 %
2008	Haihua Shen	*Coverage Directed Test Generation: Godson Experience*	Godson1 processor	92 %
2013	Kelson Gent and Michael S. Hsiao	*Functional Test Generation at the RTL Using Swarm Intelligence and Bounded Model Checking*	ITC99 benchmark	90-100 %

2.4 Bayesian Networks and Markov Model approaches

The Bayesian networks are probabilistic graphical models. In this case, each node represents a random variable and each edge between nodes represents the probabilistic dependencies among the random variables.

These networks can express the joint probabilistic distribution compactly between variables

and can express the conditional independence. These combine the principles from graph theory, computer science, and statistics. Moreover, the Bayesian Networks contain conditional parameter and network structure.

We define the number of nodes as n, and $parent_{X_i}$ are the set of X_i's parents, then the conditional probabilistic distribution of X_i could be defined as $p(X_i|parent(X_i))$. So the probabilistic distribution of the Bayesian networks is defined as:

$$p(X) = \prod_{i=1}^{n} p(X_i|X_1, X_2, ..., X_{i-1}) = \prod_{i=1}^{n} (p(X_i|parent(X_i)) \tag{2.1}$$

The Bayesian networks have been applied into this topic and the works published show good results. However, manual configuration is necessary to perform the functional verification.

In 2003, Shai Fine and Avi Ziv [22] published the Coverage Directed Test Generation for Functional Verification using Bayesian Networks. They experimented with a model of a PowerPC processor and a Storage Control Element (SCE) of an IBM z-series system. They concluded that the method provides the ability to perform functional verification with feedback (CDG), as well as the ability to cover hard cases and improve the coverage rate of progress. However, some manual configuration was necessary for perform the verification.

On the other hand, Markov models are also approaches that could be used to do the coverage of digital systems, which were introduced by Andrey Andreyevich Markov. These models are statistic methods which use probability measurements for sequential models of the represented data by vector sequences. In 2007, Ilya Wagner, Valeria Bertacco and Todd Austin [23] proposed a simulation technique of closed loop for hardware verification. They presented a tool named StressTest, which was based on random instructions generator by means of a Markov directed model with activity monitors. They used two micro-architectures for the developed experiments and proved that StressTest found more bugs with less effort than generation techniques of random probes.

Other work carried out is [24] where Jian Wang, Huawei Li, et al. proposed a method based on Markov analysis and a finite state machine models. Also, the model abstraction is made in a manual way. Due to this, the fidelity of the model depends of the human intervention. They used some benchmark circuits using the proposed method. According to the results, they concluded that the proposed method was more efficient than constrained random simulation in abstract state space exploring. Also, they mentioned that guided simulation has much better efficiency in target states exercising than the traditional abstract distances guided simulation.

2.5 Methods based on mutations

This technique was originally proposed in the software design where a software program is syntactically modified. The mutation analysis technique consists of the systematic application of faults to the original hardware description (e.j. VHDL, Verilog description). The hypothesis is based on the original hardware implementation because if this implementation is correct then one or more faults applied could be detected during the functional verification process.

A mutation is considered killed when it is detected by the insertion of a test sequence during the verification process, after that, the test sequence is saved into the test suite and is considered efficient. Sometimes, an applied fault does not produce a different behavior in the device then, this fault is considered equivalent mutation. These cases need to be considered in the verification plan, to avoid waste in the simulation process.

An example of fault insertion is shown in the next code where the or operator is changed by the and operator.

Listing 2.1. Example of fault injection

```
//Before fault injection
D <= (A and B) or C
//After fault injection: or <- and
D <= (A and B) and C
```

There are different works using mutation analysis to search bugs in designs, however, one problem with this technique is that a lot of time is require if all faults are evaluated.

A mutation is a single fault injected into a design copy. For each test case, the mutant is executed after the simulation of the original design and both simulation results are compared. When the simulation is performed if there is any difference at the design output, then this test is capable of killing the mutant. A huge database of mutants by applying different predefined fault injections is obtained. The number of mutations becomes the coverage metric. The analysis is a form to measure the quality of test data and show the main characteristics for their capability of simulating potential design errors and the propagation of the erroneous behavior to some monitoring points.

In 2011, Jorge Tonfat, Gustavo Neuberger and Ricardo Reis [25], proposed a methodology based on the verification methodology manual (VMM). This methodology was used for the verification of logic modules in a Gigabit Ethernet Switch. The verification environment was

composed of some modules which may be reused, for the functional verification of other devices. The results obtained showed high coverage percentages of the device under verification. The obtained values were greater than ninety percent.

Also, in 2011, Xiaoke Qin and Prabhat Mishra [29], proposed a technique that exploits the structural similarity within the same bound as well as between different bounds. They concluded that their approach saved time to rediscover the same knowledge for each core, without the overhead of forwarding too many conflicting clauses. They said that their approach is smaller (2-10 times) compared to existing methods. They used the information from the first core for test the next cores in a multi-core architecture.

Different works have been published using techniques based on mutations [26–30]. In 2012, Tao Xie, Wolfgang Mueller and Florian Letombe [30] proposed a verification methodology based on the generation of mutations in the device under verification (implementation). In this case a mutation was defined as a modification or alteration of the design code. These mutations served as a metric to guide the coverage progress during the simulation. For each test case, the mutations were simulated after the original design and both results are compared. The tests were performed with a microprocessor soft MB-LITE. The results obtained with this methodology reported that the tests were verified in the majority, with an average value in the number of iterations. One problem is that a lot of simulations can be used to evaluate the mutations and there are redundant mutations that increase the simulation time.

2.6 Methods based on data mining

Data mining has been applied to design some verification methods. This area has shown good results in different methods according to researches. Also, some works on data mining have been applied to design verification [31–34]. For example, Shobha Vasudevan, David Sheridan, et al. in their work [35] proposed a methodology based on a data mining algorithm, which uses a decision tree based on a supervised learning algorithm. Before their assertions are entered, these are passed through a formal verification engine to filter out the spurious candidates. They presented GoldMine, which is a tool for automatically generating RTL assertions. The method proposed was divided into two spaces; one static and the other dynamic techniques. The static analysis techniques were used to direct the data mining process. And the dynamic technique was used to simulate the device and obtain the coverage results. Some their contributions are as follows:

- Their method reduced the validation phase by distilling random stimuli and achieves coverage of unexplored spaces earlier than typical in the design cycle.
- Their GoldMine tool was applied in the OpenSparc T2 processor.
- They introduced an algorithmic design methodology based on the combination of statistical, dynamic methods with deterministic, static methods.

They used the input coverage space to evaluate the coverage of an assertion. They said that an assertion covers an entry in such a truth table if the values of the features are met. The input coverage space of an assertion refers to the percentage of truth table entries covered by that assertion. They did to mention that though methodologies compute the RTL statement coverage of an assertion, none have been implemented practically, due to using input coverage space to evaluate the coverage of an assertion.

Another work is [31] where Samuel Hertz, David Sheridan and Shobha Vasudevan presented a method generating assertions automatically in hardware devices. This method involves a combination of data mining and analysis of the register transfer level (RTL) design. Their methodology uses a combination of data mining and static analysis. The experiments were development using OpenSparc T2 processor. Their methodology is composed of:

- Static analysis
- Data generator
- A-MINER
- Formal verification
- A-VAL Evaluation and Ranking

They introduced an algorithmic methodology design subjectivity into the assertion generation process. The combination of statistical, dynamic methods was novel in the context of assertion generation. Their method abridged the validation phase by distilling random stimuli and achieved coverage of unexplored spaces earlier than typical in the design cycle.

On the other hand, in the work [32], When Chen, Li-Chung Wang, et al., proposed a methodology of knowledge extraction from constrained-random simulation data. The extracted knowledge then was reused for two purposes:

- For producing more tests similar to those important ones
- For producing new important tests that, for example, can activate assertions not covered before.

Their methodology begins by extracting a list of conditions from the assertions for monitoring. A novel test with respect to these conditions is identified in the simulation. The

extracted knowledge means the rules describing the special properties of the novel tests.

This methodology was based on the representation of programs in multiple snippets of instruction sequences of equal length k where k is user-supplied input. Their work proposed a learning methodology to extract knowledge from simulation in constrained-random processor verification. They used a dual-thread low-power 64-bit Power Architecture-based processor core.

2.7 Functional Test Generation based on extraction of State Machines

According to [36] the metrics based on finite state machines use coverage of states, transitions or paths in a representation of finite state machines (FSM). Some control portions are represent better by a FSM's collections interacting between them. In this case, we can use metrics defined by multiple state machines. For example, the metric of pars of arcs needs to exercise all possible pars of transitions for each FSM's controller par. The FSM's can be classify in two categories:

- Hand-written is a FSM's which captures the behavior of the design at a high level.
- Finite State Machines extracted in an automated way from the design description. Typically, after the set of state variables is chosen, the design is mapped with this set in order to obtain an abstract FSM.

The metrics in the first category are less independent from the implementation details. The variables of state in the metrics for the second category can be chosen in a manual way by mean heuristics. Generally, the small processors have a big number of paths, but due to the simulation cost is small then it is tolerable.

When the amount of details in the finite state machines (FSMs) is increased then the precision of the coverage metric is increased, but it produces an increase of complexity in the interpretation of the coverage data.

In the case of designs which have a big amount of concurrent control where few iterations of a FSM's can produce a more difficult search of bugs, there are simple concerning metrics. A set of test which covers all transitions of all possible states of control, can maximize the probability of finding errors in the design while minimizing the simulation time. It is necessary to distinguish between the data and control to extract the control part of a design.

In [37] Iuseppe Di Guglielmo, Luigi Di Guglielmo, et al. proposed a methodology of functional coverage based on the generation of finite exceeded state machines. The results obtained are compared with other generation methods based on genetic algorithms and pseudo-deterministic methods.

2.8 Methods based on Theorem Proving

Theorem proving is a technique which begins with a specific goal and then it split this goal in two or more sub-goals. These sub-goals are split again building proof trees. The proof trees can be solved by: lemmas, axioms, decision procedures, trajectory-evaluation runs among other techniques. The proof is complete when all conditions are verified.

The methods based on Theorem proving use a model of the digital system which is a set of mathematical definitions in mathematical formal logic. The properties of the system are mapped like theorems. These theorems are generated based on these definitions. The classic version of theorem proving methods is based on high logic variants of high order. Software tools as HOL, PVS and ISABELLE/HOL have been developed in order to perform the functional verification based on Theorem proving.

Theorem proving uses a constructive logic. An advantage consists of the variation of a probe which produces an executable version of the algorithm that has been verified. A disadvantage is the constructive probes are stronger than classic probes.

Another advantage of these type of techniques is the methods can test complex systems while it does not review directly each one, each state, while the logic is commonly more expressive. A disadvantage is that it needs the intervention of humans and creativity in order to finish the probes increasing the verification time.

There are different works [38–41] using theorem proving in order to perform the functional verification of digital systems. For example in [38], Mark D. Aagaard, Robert B. Jones and Carl-Johan H. Seger presented the verification of an instruction-length marker (the IM) against an implementation-independent specification of IA-32 instruction lengths. That was performed by mean a combination between model-checking and theorem proving techniques. The theorem proving guided the decomposition in different tasks into the limits of model-checking. They concluded their method founds new bugs in the design and it improved the original specification.

On the other hand, in [40] John O'Leary and Roope Kaivola presented an approach for verification based on symbolic simulation (Relational STE). The method used a descomposition

in a logical level by means theorem proving. The method was added into a software tool in order to outperform the symbolic simulation. They use multipliers as devices under verification concluding the proposed method is a good alternative improving formal verification.

Also, in [39] Roope Kaivola and Mark D. Aagaard proposed the combination of model-checking and theorem proving in order to verify a floating-point divider unit of an Intel IA-32 microprocessor. They used a software tool which support model-checking and theorem proving techniques. They concluded the main advantages of their approach are: the safety of a mechanically verified proof combined with the freedom of complete control in order to perform the verification of a complex device.

2.9 Methods based on Model Checking

Model checking technique was introduced by Edmund M. Clarke and E. Allen Emerson [42]. The technique consists in verifying that the logical conditions follow to be true through all reached states from the Device Under Verification (DUV). If a propriety is kept then Model checking produces an opposite-example, it means, an execution path which produces a state where the propriety is false. Model checking can be used to verify if a set of specified functions is met using a temporal logic language with respect to system behavior model.

Definition 1 ***Model checking*** *is an automated technique that given a finite state model of a system and a formal propriety, this reviews in a systematic way if a propriety is met for that model. This technique is based on temporal logic. Temporal logic idea means an equation which is not static true or false in the model as it is in the propositional and predicative logic. Also, temporal logic models contain different states and an equation can be true in some cases and false in other [43].*

A lot of dynamic systems contain components of state which change through time. For example, sequential circuits, which contain flip-flops and latches. Also, in this technique, the models are defined as M which are transition systems and proprieties as ϕ which are equations in temporal logic.

Model Checking can be specified in two ways:

- **Model Checking global problem.** Given a model of a finite structure, M, and an equation,ϕ, a set of states is determinate in M which satisfy ϕ.

- **Model Checking local problem.** Given a model of finite structure, M, an equation, ϕ, and a state "s" in M determines if "s" satisfies ϕ.

Where: The outputs consist in a "YES" if M $\models \phi$ and "NO" in the other case. In the second case, also, a path in the system behavior is executed, which produces the fault. This automated generation of paths is a main tool in the design and the detection of faults of the system.

The objective of model checking is to perform the functional verification of the sequential proprieties of a dynamic system. A dynamic system has a component of state, which changes through time. The typical systems, are sequential circuits which contains delay elements such as flip-flops and latches [44].

Different works have been carried out based on Model checking technique. The investigators, Javanand Asok Kumar and Ahobba Shobba Vasudevan in [45] proposed a statistic method using Model Checking in order to perform the verification of a schema for the multi-core processor management. They used this technique in order to complement the results obtained during the simulation of the device with statistical results obtained by means of that technique. They considered a set of properties to be verified based on the characteristics of time-out and the adaptive power gain. In the results the obtained percentages were over 80 % and 90 % for different modules.

On the other hand, the researchers Ali Alphan and Sharad Malik in [46] proposed a method based on the validation on run time of a device using Model Checking. They considered the use of a hardware on-chip in order to detect bugs using hardware sentences. According to obtained results using this hybrid method, they concluded that this technique obtains sentences which help to Model Checking in order to obtain good results on run time verification.

Other works were based on the Model Checking method [47–53]. These methods use Boolean satisfiability procedures (SAT procedures) or compact representations of Boolean functions such as binary decision diagrams. The algorithms determine the states that satisfy a model formula by a graph-theoretic analysis of the space (kripke structure). One problem of these algorithms is caused when the size of the state graph grows exponentially with the number of parallel components in the digital system.

2.10 Resume

In this chapter, we described the importance of the functional verification by the industry. Also, different methods have been proposed in order to perform the Directed Coverage test generation (CDG). This technique has represented an important technique for the test of digital systems in real applications.

Among the carried out works, there are some works where the Artificial techniques have been used. The main used techniques in the functional verification are: Genetic algorithms, Bayesian networks, Markov models, Data mining, Mutations, etc. Every technique has obtained good results. Moreover, some proposed techniques have reached high functional coverage percentages using different devices. Although, one disadvantage is that most of these methods require a lot of computer resources. According the researches these methods were better than the pseudo-random test generation method.

There are some works using evolutive algorithms as Genetic algorithms and swarm intelligence. According the authors the methods using these meta-heuristic algorithms have produce better results than the generation based on pseudo-random generation. Also, we can mention that there have been proposed new meta-heuristics as Particle Swarm Optimization algorithm, Differential Evolution algorithm among others. Due to this, we propose the use of these meta-heuristics in order to find test vector sequences to maximize the functional coverage obtained from the verification of the devices. In the next chapter, the main definitions about functional verification will be presented, also, the formal definition of coverage model, directed functional verification, coverage metrics, etc. will be described.

CHAPTER III

Digital Design and Functional Verification of Digital Systems

Chapter 3

Digital Design and Functional Verification of Digital Systems

Different devices such as smart-phones, tablets, lap-tops, smart-watches, drones, etc. contain chips with millions of transistors on small encapsulates. According to the Law of Moore every eighteen months the digital circuits contain twice the number of transistors, therefore, the complexity of their design and functionality is increasing. The design of digital systems needs to meet the necessary and requirements according to the current technology advances.

Functional Verification is one of most relevant steps during the design of digital systems. Different from the errors in the software, the errors of hardware are more expensive because the hardware devices need to be physically replaced. Due to the constant increase of functionality complexity in digital systems new verification methods have been proposed. Moreover, new software tools and platforms are used in order to reduce the time used to perform the verification of devices. Compatibility and scalability are new challenges produced by the new capabilities of the digital systems.

Different elements are used in the Functional Verification area. Moreover, the behavior of a Digital System needs to be modeled in order to measure what portion is exercised during the Functional Verification process. Different software tools are employed in order to model the functional verification process for a device. Also, the engineers review if the correct behavior is met based on a representation which allows them to analyze the responses of the device in different conditions.

Figure 3.1 represents a diagram which contains three fundamental elements in the design process of digital systems, which are: the design intention, specification and implementation.

The main objective of Functional Verification is to overlap these three elements because if this is achieved, the specification will be contained into the design intention, and finally, the intention will be contained in the implementation of the device [1].

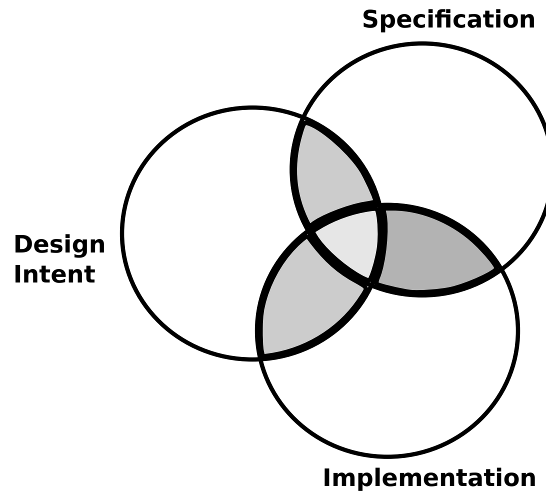


Figure 3.1. Design Intention diagram.

In this chapter, different elements about functional verification will be explained. For example, the main definitions as digital system, functional coverage, functional verification, coverage model, among others will be explained. Also, the design, modeling and simulation of digital systems are described in order to perform the functional verification process.

3.1 Digital systems design

Currently the design of digital circuits represents an art because it involves human intention. Therefore, the representation of the functionality of digital systems can be mapped in different implementations according to human criteria. The first step to design a digital system consists in organizing the requirements and characteristics which are needed to translate all characteristics to a physical device. Different abstraction levels are used in order to represent the digital circuit. In all abstraction levels, the intention is to improve the design phase reducing the effort during that process.

An important aspect is the definition of Boolean functions which constitute the main elements of digital systems. A Boolean function can be defined as follows:

Definition 2 *A Boolean function is a function* $f : \{0,1\}^m \rightarrow \{0,1\}$ *for some* $m \in \mathbb{N}$.

A digital circuit can be described as a graphical representation of propositional formulas where common sub-formulas may be re-used by allowing what we will call fan-out greater than one. A digital circuit can be defined based on a set of Boolean functions as follows:

Definition 3 **A digital circuit** *can be described as a set of Boolean functions which receives a set of input sequences* $I = \{i_0, i_1, ..., i_n\}$ *to produce a set* $O = \{o_0, o_1, ..., o_m\}$ *of output sequences. Where* i_j *is the* j_{th} *input data sequence accepted by the device and* o_k *is the* k_{th} *output data sequence produced by this.*

A sequential digital circuit can perform one or more operations every clock cycle, even if it keeps in the same state as the previous cycle. Basically, there are two types of circuits: combinatorial and sequential circuits. The first type consists in circuits without clock signal. The information is processed immediately when a test sequence in injected at the input. The second type consists of circuits with one or more clock signals. The information is processed based on the clock cycles. The sequential circuits can be split in synchronous and asynchronous. In the fist case a sequential synchronous design can be defined as follows:

Definition 4 *A sequential synchronous hardware design can be represented as a finite state machine* $M = (I, O, S, t, \delta, \lambda, n, m)$ *where:*
n = number of Boolean inputs.
m = number of Boolean outputs.
t = number of Boolean state variables.
I = input space of 2^n *inputs.*
0 = output space of 2^m *outputs.*
S = state space of 2^t *states*
δ*:* $S \times I \rightarrow S$ *is the next state of the system.*
$\lambda : S \times I \rightarrow O$ *is the output function.*

Taking account the last definitions in general way a digital system can be defined as follows:

Definition 5 *A digital system is a set of devices for the generation, transmission, processing or storage of digital signals. This is a combination of devices designed to manipulate physical quantities or information that are represented in digital form; that is, they can only take discrete values.*

Other main term into the functional verification refers to the device behavior. A definition is as follow:

Definition 6 The device behavior *is composed of a set of logic functions* $f : \Gamma^n \rightarrow \Theta^m$ *where:*
Γ^n *is the Cartesian product :* $i_0 \times i_2 \times ... \times i_n$
Θ^m *is the Cartesian product:* $o_0 \times o_1 \times ... \times o_m$

On the other hand, a digital system can be modeled using different abstraction levels. There are four abstraction levels in order to represent and model a digital circuit.

- Structural representation. Is the representation with all signals which constitute the device. It is not necessary to specify the functionality of all modules because the objective is to present the different signals and blocks of the device.
- Functional representation. Consists of the functionality of the device. It can be described as the black box representation of the circuit with input and output signals.
- Physical representation. The physical characteristics such as weight, size, temperature, etc.

In order to perform the functional verification of a device it is necessary to make the implementation of a digital system in a hardware description language i.e. VHDL, SystemVerilog, Verilog. A hardware model means the implementation of a digital system specification.

Digital systems modeling involves a set of steps which allow the implementation of a device. These steps are:

- Establish specifications.
- Define inputs and outputs.
- Capture the design (HDL schematic).
- Syntax verification.
- Model syntax (translate to an equivalent of logic gates).
- Functional simulation.
- Mapping (Becomes the equivalent of the logic gates to device resources to be programmed. Backpropagation).
- Place and Route (placement, routing, internal interconnection).
- Simulation times.
- Configuration.
- Depuration.

Figure 3.2 shows the main steps for the manufacture of a digital circuit. First, the design engineer reviews the requirements and provides a specification. Then, the device is in high level modeling, that is, at a black box level describing inputs and outputs to perform the functions required in the specification. The next step is to capture a schematic design using a design tool and a Hardware Description Language (HDL). Once the implementation of the device is completed, verification engineers review the implementation and the functional specification. This process is conducted in order to produce a verification plan. After a verification plan is proposed, the verification problem is defined and a set of elements (coverage metrics, verification environment, test sequences) is proposed for verification. After the implementation of the device is completed, the device is mapped, placed and routed. After, the component is packaged and tested physically. In the next section the main elements of functional verification will be described.

3.2 Main elements of Functional Verification

There are different elements involved in the functional verification process. First, the verification engineers need to review the specification. This document contains the requirements and constraints of the device. Also, a verification plan is designed based on the characteristics of the implementation and the specification. This plan involves what and how will be verified during the verification process. There are different aspects of the verification plan some of them are: the coverage measurement, the response checking and the stimulus generation. The coverage measurement defines the scope of the verification problem. It means, the type of coverage metric, the coverage goals and events to be covered. The response checking describes how the responses of the device will be compared with the specified responses. The stimulus generation consists of the set of techniques to be used in order to generate the test sequences.

The assertions represent conditions that must be met. Unlike software verification, where only the assertions are then verified, in hardware designs these assertions need to be accomplished in a specific time. Due to this, it is necessary to implement the design in a temporal language.

Figure. 3.3 shows the flow diagram of a methodology for performing the functional verification process. This functional coverage methodology starts by reviewing the digital system specification after analyzing the design implementation or device. Based on the above, a functional coverage model is proposed. Given that model, it is necessary to propose a

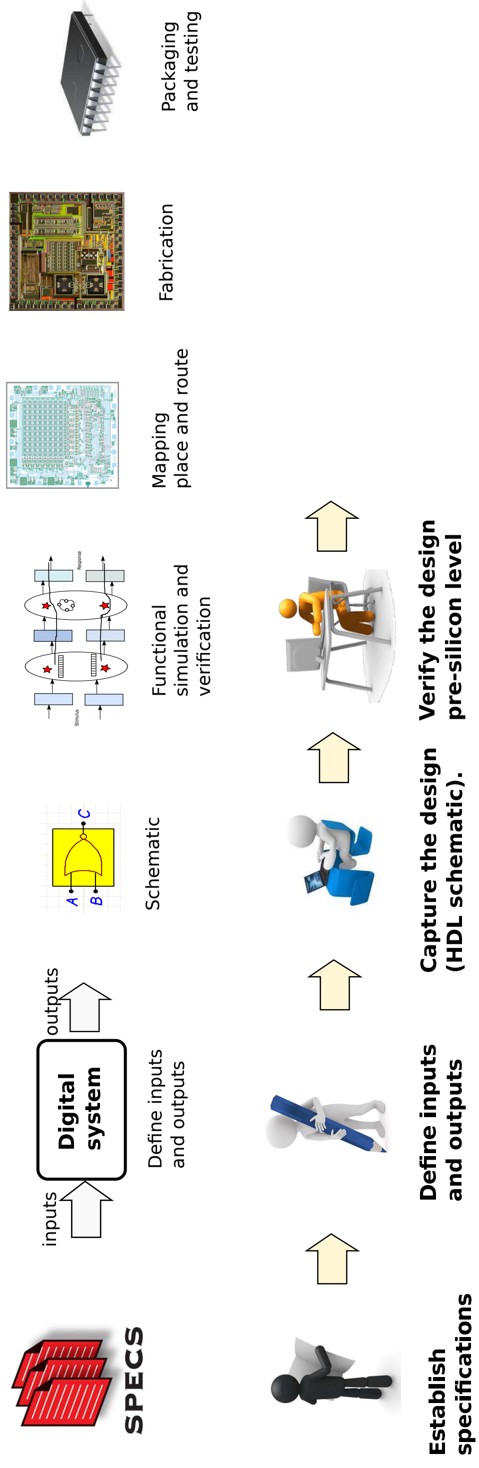


Figure 3.2. Main steps of digital system design.

verification plan. From this plan the verification engineers perform the functional verification of the digital design. Once the simulation stage is over, the results are reviewed and if a desired percentage was reached, the process finishes and if not, the verification tests are modified and the simulation is performed once again.

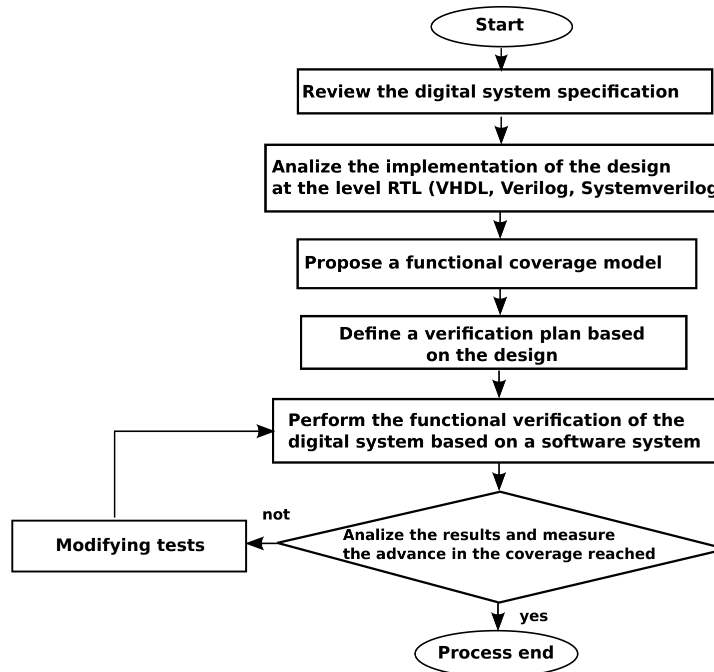


Figure 3.3. General methodology for the functional verification process.

3.2.1 Functional Coverage

Coverage is the measure of the integrity of set of test. It is important to differentiate the functional coverage and manufacturing fault coverage, which is used to assess the number of faults in the manufacture of integrated circuit.

A definition of coverage is the number of goals achieved by the test set applied. Generally, the coverage is expressed as the percentage of defined goals that are concretized. For example, the percentage of sentences exercised from the RTL hardware description.

Functional coverage is based on the specification and the type of coverage metrics to be recorded. This is used in order to measure the relevant or interest characteristics of the Device Under Verification (DUV). For instance, to measure an address of 32 bits could generate 4

billions of values. However, there are values numerically different but functionally equivalent. Therefore, the representation of these values by mean coverage points can be made based on the functionality. This strategy can reduce the search space, then reducing the verification time.

Functional coverage can be defined in terms of the device behavior as follows:

Definition 7 *Functional Coverage represents the region of the device behavior which is exercised by the stimulus at the input. It means the percentage of the verification goals which have been met.*

Functional coverage is responsible for defining the specification and implementation of metrics to be remembered.

Another definition of coverage is defined as the percentage of verification objectives that have been met. It is used as a metric for evaluating the progress of a project verification in order to reduce the number of simulation cycles expected in verifying a design.

A purpose in identifying the functional coverage remains to be done. During the analysis, the tool that report the functional coverage can compare the number of containers that store at least one sample against the total number of containers. If a container has at least one sample, it is known as the functional coverage gap. Using Hardware Description Languages (HDL) like SystemVerilog different characteristics of the implementation can be monitored, some of them are:

1.- Coverage variables and expressions , as well as cross coverage.
2.- Bins automatic and defined by the user.
3.- Bins associated with sets of values or cross product transitions.
4.- Conditions filtering on multiple levels.
5.- Events and sequences to trigger automatic coverage.
6.- Activation and procedural query coverage.
7.- Optional directives to control and regulate the coverage.

A goal of the functional coverage consists in identifying what needs to be done. During the analysis, the tool which produces the coverage report can compare the number of containers which save almost one sample regarding the total number of containers. One container which has not one sample is named as a coverage hole.

Figure 3.4 presents the main trends in functional verification techniques. In this case, functional coverage is one of more used parameters for functional verification.

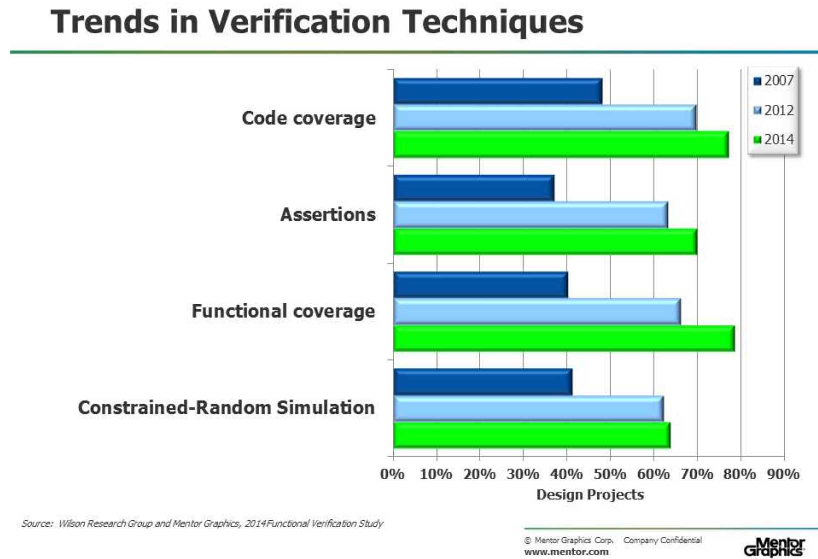


Figure 3.4. Trends in Functional Verification Techniques.

3.2.2 Coverage Points

A coverage point is the representation of the values of a variable in a coverage model. The formal definition is as follows:

Definition 8 *"Given a device implementation γ based on β_i characteristics and a functional specification φ, a coverage point is the set of possible values $b_0, b_1, ..., b_k$ by every characteristic $\beta_i \in \gamma$".*

A coverage point can be defined as the sign of a scalar value or expression. It can be a variable or integral expression. Each point includes a set of *bins* associated with sampled values or the values of the transitions. The main goal of a coverage point is to ensure that all interesting and relevant values are observed in sampled expression. Examples of coverage points are: packet length, instruction code, interrupt level, bus transaction, completion status, buffer occupancy, demand patterns bus. The coverage points can be grouped in coverage groups. A coverage group contains the set of coverage points and other elements among which are:

- **Clock event.** Defines the event when coverage points are sampled. If the clock event is omitted, users must proceed.

- **Coverage points.** Can be a variable or an expression in the implementation.
- **Crossover coverage.** Is a relation between two or more coverage points or variables.
- **Coverage options.** Are used in order to control the group behavior.
- **Optional formal arguments.** It means the arguments which are mapped when an instance of coverage group is generated.

When a coverage point is in use, the goal consists in observing different values of the above mentioned point. Due to this, Hardware Verification Languages like SystemVerilog can create automatically fields known as *bins*. For example, when the mechanism of automatic creation of *bins* is used, SystemVerilog create N *bins* to store the sampled values of a coverage point. The N value is determined as follows:

- For a coverage point, N is the cardinality of the numeration
- For some other point of integral coverage, N is the minimum of 2^M values and the value of the option $auto_bin_max$, where M is the number of bits required to represent the coverage point.

If the number of automatic *bins* is less than the number of possible values $N < 2^M$, then 2^M values are uniformly distributed in the N bits. If the number of values, 2^M, is not divisible by N, then the last *bin* might include the rest of the (N-1) additional items. For instance if M is 3 and N is 3, then the eight possible values are distributed as follows: $< 0 : 1 >, < 2 : 3 >, < 4, 5, 6, 7 >$. These properties of Hardware Verification Languages can help to make more efficient verification environments.

The coverage holes can be seen as correct tasks which have not been checked in the simulation. It is necessary to analyze that the generated holes are valid tasks, since in the case they are not valid tasks, it is necessary to adjust the restrictions introduced inside the model to eliminate that holes. When incorrect tasks are generated and have not been covered (false holes), it reflects problems in the coverage model which is used.

After the device simulation is made the analysis of generated holes is required. The coverage information shows which tasks have been covered and which are not covered even.

Some points have common tasks, therefore, it is possible to group them in order to reduce the number of holes and thus to direct the vectors generation to cover the holes [54].

3.2.3 Functional Coverage Metrics

Coverage metrics ensure optimal use of resources simulation, measurement validation, and the direct simulation to design areas. Due to the need of know how much functionality is exercised coverage metrics are required in order to measure which region has been covered. A general definition of coverage metric is described as follows:

Definition 9 *A coverage metric can be defined as a heuristic to measure the portion of the device behavior that has been verified. The main objective of this measure is to reflect which parts of the functionality have been met with a correct execution during the processing of the information by the device.*

According to [36] coverage metrics help to improve the functional verification of digital systems:

- Acting as heuristic measures that quantify the full verification.
- Identifying aspects adequately trained in-leading design and generation of input stimulus.

An appropriate metric for evaluating the functional coverage of a set of tests can be a fraction of the specified behavior of a digital system that is exercised. That is, the fraction of errors detected by that set. Finding all possible tasks has an exponential complexity and trying to prove each one may require huge computing resources. Another metric can achieve fraction of states or transitions that have been exercised.

Usually the designers use a set of metrics to measure the progress of simulation based on validation, starting with simple metrics that require little effort and gradually use more sophisticated and expensive metrics. A formal way for more sophisticated simulations also creates a difficulty. For instance; if a metric M_1 involves the M_2 metric, then the input stimulus reaching the full coverage M_1 does not necessarily improve the error detection (*bugs*) that the input stimulus reaches to complete the coverage M_2. (The metric M_1 involves the metric M_2 if and only if, when in any design the set of input stimulus S reaches 100% of coverage M_1, S also reaches 100% of coverage M_2).

One of the main uses of coverage analysis is to measure the efforts of the adequate and progressive validation. The direct correspondence between the coverage metrics and error classes should ensure that full coverage could detect all errors of a certain type with respect to the metric.

The level of confidence is directly proportional to the quality of the coverage metric. In most of cases, the coverage metrics inform how many tasks were exercised by the test cases saving the

number of changes of the characteristics which describe the exercised proportion of the device behavior. It means, when more information is reflected from the implementation through the coverage metric, then, more confidence is obtained about how well the device functionality was verified.

According to the specification and the implementation the coverage metrics can be divided in two different categories: implicit and explicit metrics [5]. The first case consists of coverage metrics based on the features of the design implementation (using hardware description languages: VHDL, Verilog, SystemVerilog) for example the code coverage metric. The second case consists of metrics based on the functional specification and the characteristics of the behavior, for example the functional coverage metric.

There are differences between the implicit and explicit metrics, for example the next set of code represents a set of instructions of a CPU. In the code the RST instruction is not enabled into the options. In this case a code coverage report can tell us that 100 % was reached. However, the functional coverage metric should report different results because one variable was not exercised during the verification.

Listing 3.1. Example of coding error undetectable by code coverage

```
enum{ADD, SUB, JMP, RTS, NOP} opcode;
...
case (opcode)
        ADD: ...
        SUB: ...
        JMP: ...
        default: ...
endcase
```

The last example shows that the functional characteristics depend on the functional specification and not of the device implementation. The differences must be understood and used together to improve the confidence of the verification process. In the next section some main concepts about how the obtained information from the device is managed will be described.

There are different coverage metrics among which are:

- Statement coverage: This metric is known as block coverage because it measures a set of lines which is executed in a block of code in the implementation. The visual representation can depend of the software tool which is used.

- Expression coverage: Measures how many expressions are executed into the implementation code.
- Paths coverage: Measures the different ways to execute a sequence of statements. Different from statement coverage the path coverage can detect if a block of code was exercised using only a subset of the all possible conditions to execute it.
- Coverage of mutations: This is a measure of the number of changes of the different conditions into the code which are executed in the implementation.
- Functional coverage: is a metric based on the functional specification. This metric measures the number of characteristics of behavior which are exercised. Different from metrics based on implementation functional coverage shows if a specific part of behavior has been not exercised. In order to use the coverage metric a functional coverage model needs to be proposed.
- Code coverage: This type of coverage measures the number of lines of code which have been executed during the simulation of a digital system. It is implicit from the implementation and it is named as implementation coverage. The main objective of this coverage metric is to show which portion of code has not been exercise.
- Coverage Discounting: This type of metric combines mutation and functional coverage metrics in order to discover which points actually have not been covered

3.2.4 Calculation of Functional Coverage

The cumulative coverage considers the distribution of all instances of a particular item or coverage point. In contrast, the coverage of an instance means the specified coverage of the specific coverage instance in which is focused.

To make the calculation of coverage for a coverage point, we must first determine the total number of possible values, also known as the domain. There may be a value *bin* or multiple values *bins*. Coverage is the number of sampled values split by the number of *bins* in the domain. For instance, a coverage point can be a variable of three bits which has a domain of [0:7] and is normally divided into 8 *bins*. If during the simulation the involved values to 7 *bins*, are sampled, the report will show 7/8 or 87.5% coverage for this point. All these points are combined to show the coverage of the whole group, and then all groups are combined to generate a coverage percentage for all simulation databases.

This is the state of a single simulation, so we need to store the coverage over time. We must

take care for changes, and we can see further simulations or add new restrictions or tests. Eq. 3.1 describes the calculation of functional coverage for a set of points. The coverage is the division between the sum of the product of coverage C_i for each point with the weigh W_i of each one divided by the sum of all weights.

$$Cg = \frac{\sum_i W_i * C_i}{\sum_i W_i} \tag{3.1}$$

This is the state of a single simulation, so we need to store coverage over time. Watch for changes, and we can see further simulations or add new restrictions or tests.
The calculation of coverage for a coverage point depends on whether the *bins* are explicitly defined by user or automatically created by the tool. For the *bin* defined by user, the coverage of a point is calculated as follows:

$$C_i = \frac{|bins_{covered}|}{|bins|} \tag{3.2}$$

where:
$|bins|$ Is the cardinality of the set of defined bins.
$bins_{covered}$ Is the cardinality of the covered *bins*, it means the subset of all *bins* (defined) which are covered.
For the *bins* generated automatically, the coverage of a coverage point is calculated as follows:

$$C_i = \frac{|bins_{covered}|}{MIN(auto_bin_max, 2^M)} \tag{3.3}$$

where:
$bins_{covered}$ Is the cardinality of the covered *bins*, the subset of all *bins* (self-defined) that are covered.
M : Is the minimum number of bits needed to represent a coverage point (Coverpoint).
$auto_bin_max$ Is the value of the option $auto_bin_max$ in SystemVerilog.

It is important to understand that the union considers cumulative coverage of all significant bits; this includes the contribution of all *bins* (including the *bins* overlapped) of all instances).

To determine if a particular *bin* of a coverage group is covered, the calculation of the cumulative coverage considers the value of at_least option for all instances that are covered.

Due to this, a *bin* is not considered covered unless the amount "hit count" equals or exceeds the maximum values at_least of all instances. Using the maximum value that represents the most kept selection.

3.2.5 Functional Coverage Model description

A functional coverage model represents the initial step into the functional verification. It is defined as a coverage space representing the interrelationships between the inputs, outputs and components of a device [1]. Independently of the method, when the functional verification process is performed, a coverage model is used. The coverage model is based on a type of coverage structure or coverage metric, i.e. finite state machine (FSM), statement, branch path, and expression coverage, etc. The model is a fundamental piece of the verification method representing a golden model to describe the device behavior.

A coverage model consists of tasks, events, conditions, etc. which capture the Device Under Verification behavior. In other words, it is an abstract representation of the device behavior composed by attributes, features and their interrelations. In this research we define a coverage model as follows:

Definition 10 *"Given a functional specification φ, the coverage model is defined as the set of features ζ which has a set of used constraints τ representing the full device behavior".*

where:

$$\zeta = \{\beta_1, \beta_2, \beta_3 ... \beta_n\} \tag{3.4}$$

$$\beta_i = \{b_0, b_1, ..., b_k\} \tag{3.5}$$

Each b_i is a relevant feature value of a coverage point β_i and the set of constraints:

$$\tau = \{p_0, p_1, ..., p_t\} \tag{3.6}$$

Where p_i is a possible valid set of values for each b_i.

Figure 3.5 shows a coverage model example for a UART-IP core device. The set of points represents the abstract way to represent the behavior. In general, a coverage point can be a scalar value, an expression, an event or a condition in a digital system. An example of a coverage point

can be a command instruction code, since this is a variable which can take a set of values $(0, 1, 2, ..., k)$. Each value is known as a *bin*. Other examples of points are: an interruption level, a register input, package longitude, etc. The main objective of a coverage point is to ensure that all relevant values are represented in the proposed variable. These points are grouped in sets, which are commonly known as coverage groups.

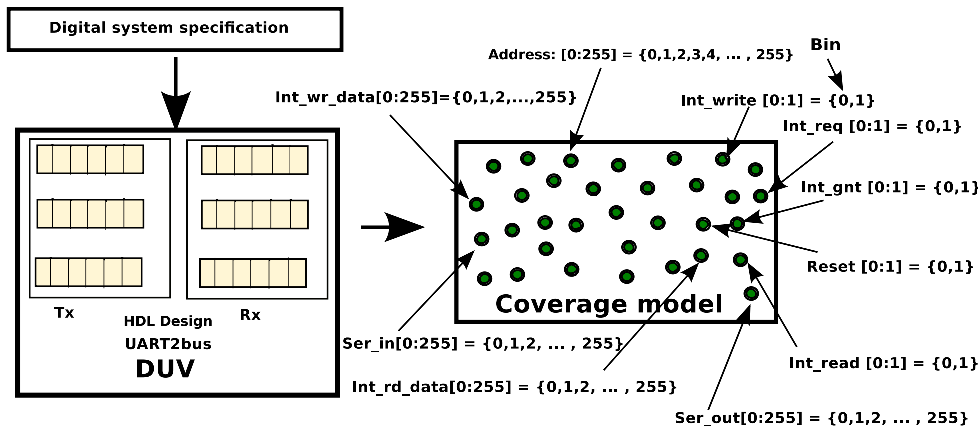


Figure 3.5. Model representation of a Device Under Verification (UART-bus) using coverage points to describe the behavior.

The coverage model contains explicit device behavior characteristics. However, this can be used with implicit coverage metrics taking the coverage information as feedback and then analyzing and evaluating the current test.

During the verification of the device behavior, a functional coverage model is proposed based on the functional specification and the device implementation. This model is designed using different levels of granularity, which means that the model can represent the original intention using a different number of characteristics. The verification engineers make a decision on the amount of fidelity by taking into account the relevant aspects of the device function and the set of values that represents the main functional regions within the functional behavior space. Moreover, the fidelity of a coverage model depends on these aspects, especially when all main characteristics are included. One problem with functional verification is that it can identify if there is a deviation between the implementation and the original intention, but it cannot ensure that there are no errors in the final implementation because the initial specification is designed by human criteria. Also, the characteristics of the device behavior, which constitute the proposed coverage model, may not be sufficient to test all device behavior.

A coverage model is a functional space coverage, which captures the behavior of a device. In other words it is a definition of a subset of the space of stimulus/response that could show an acceptable degree of confidence, that the functionality of the design is correct.

The fidelity of a coverage model is a measure of how similar the model represents the behavior of the device. If the coverage model has high-fidelity output, all device responses to stimuli applied have corresponding points or regions within the model. The functional coverage measures progress through the requirements of the device.

A coverage model contains requirements of functional verification process. The total stimuli and response space for a complex design is multi-dimensional and almost infinite. So it is unrealistic to expect the comprehensive verification of a design for all possible combinations, sequences of stimuli and responses.

For example, a solution for an adder of 32 bit is a three-dimensional space measuring a space of $2^{32} \times 2^{32} \times 2$ (Input A x Input B x carry- in). The exhaustive verification of this simple design assumes a set of input values which can be verified every nano-second and might require 1000 years . But an adequate level of confidence in the correctness of an implementation can be obtained using only a small set of peers. And that set of inputs will be the coverage model for combinational adder.

3.2.6 Directed Functional Verification for Digital Systems

The functional verification process can be described similarly as processes used in digital communication, since in digital communication original data is delivered with additional information which permits us to detect errors and correct them. In a similar way, in performing verification based on simulation, the original intended behavior (It can be seen as original information) is implemented according to functional specification and the designer criteria (It represents the additional information). Through different process steps, the behavior is verified by means of monitors, test-benches, assertions, etc. Finally, when the process finishes, the method determines whether or not the implementation meets the original proposed behavior. In a formal way the functional verification can be defined as follows:

Definition 11 *“Given a device implementation γ based on β_i characteristics and a functional specification φ, the functional verification is the process which ensures that each characteristic β_i of the specification φ is met by γ".*

Where:
$\beta_i \in \varphi$
$\varphi = \{\beta_0, \beta_1, ..., \beta_k\}$
Figure 3.6 shows the general scheme of verification process using the functional coverage as a coverage metric. In this case, at the beginning of the process, test sequences are generated in a pseudo-random way. Then different events, variables, and tasks change their initial state. However, during the process, some events are not covered, revealing coverage holes which need to be tested. Due to this, the coverage analysis is a relevant step to exercise new tasks and can be used to cover hard cases. It means that the analysis can reduce the number of repeated tasks during the verification, which reduces the simulation time.

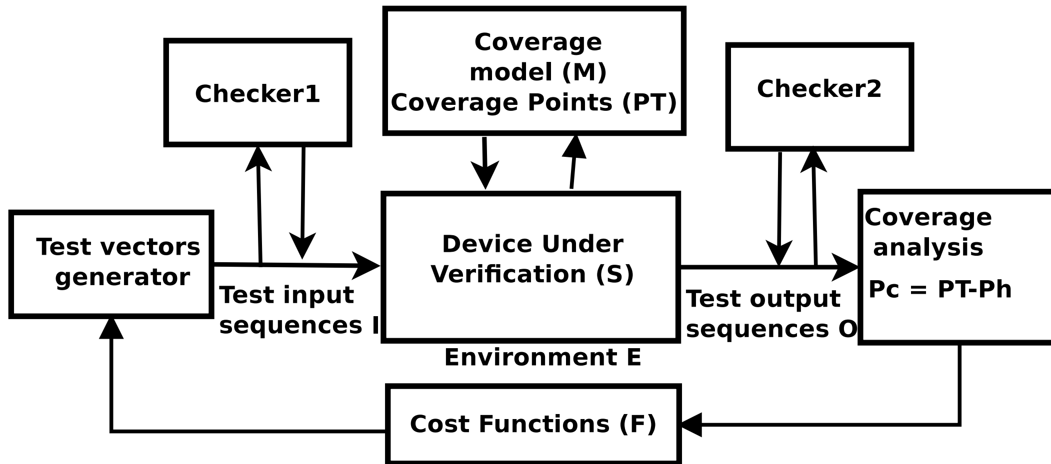


Figure 3.6. Coverage Directed Test Generation blocks diagram.

The Coverage Directed Test Generation can be described as the process which generates test sequences at the DUV input according to the feedback information during simulation. It includes different elements used to exercise the device and produce the environment interaction. In a formal way:

Definition 12 *"Given a device implementation γ and a functional specification φ, Coverage Directed Test Generation is the process which generates the test sequences $t = \{i_0, i_1, ..., i_n\} \in (2^l)^{n+1}$ exercising the features β_i of the coverage model ζ which represent the functionality of the device".*

Where:
l is the length of the input sequence.
$n + 1$ is the maximal number of sequences.
i_k is the k-th binary test sequence.
When a test sequence i_k is introduced inside the device, if a new value of $f_k(i_k)$ function of the specification is exercised, then that value and the sequence is saved into a file. After that, the set V of states is reviewed and a new test sequence is generated to test the device again. When all v_i specified states are verified, the coverage metric is reviewed [55].

When the functional verification is performed, one or more coverage models and the implementation device are connected to test the device. Then different functional tests verify which feature, also called "relevant event" (coverage point) or "variable" assigned in the model is covered and review if the functionality is working correctly. When the process finishes, the characteristics tested are reported into a file [56].

We describe the used coverage directed test generation model in this research as a 11-tuple:

$$\langle I, O, S, M, P_T, P_c, P_h, F, E, C, I_{P_c} \rangle \tag{3.7}$$

where:
I is the set of the possible input sequences $2^{\{l\}*(n+1)*}$.
O is the set of the possible output sequences o_k.
S is the Control Flow Graph describing the device behavior.
M is the proposed coverage model.
P_T is the number of coverage points.
$P_c = P_T - P_h$ is the set of the covered points in the verification process.
$P_h = P_T - P_c$ is the set of the holes (not covered points) during verification.
F is the set of cost functions $\{f_1, f_2, ..., f_n\}$ to determine how well the generated binary test sequences work.
E is the test-bench interacting between the device and the data and control signals.
C is the set of checkers used to verify the different inputs and outputs of the DUV.
I_{P_c} is the set of test sequences which exercise the maximum coverage points percentage in the M.

The actions between these elements produce the directed generation of test sequences, which can be described as follows: first, a test input sequence i_k is generated in a pseudo-random way,

and then the device implementation is evaluated. At the end of the evaluation a set of coverage information is obtained. The test vector sequences are modified according to the values of the functions f_i. These cost functions are obtained using the values of P_c and P_h at the end of each simulation respectively. P_T includes P_h and P_c, which means all coverage points. All input and output data are reviewed by the set of checkers C during the simulation. Also, the exchanged information between the coverage model M and the environment E are checked. In addition, a scoreboard module can be used in order to review whether the input and output data are correct. This is performed by capturing the data at the input of the device and the processed output. Then, the scoreboard compares the data generated from the device with the data generated based on the specification. Finally, when the stop criteria is met a test sequence I_{P_c} is obtained. It is important to mention that the coverage percentage in most of the cases is less than one hundred percent due to differing factors such as: redundant test cases, insufficient number of simulation iterations , bad checkers, low number of test sequences, lack of information from the coverage metrics, etc. Due to this, one of the challenges is to find good alternatives to test all the characteristics of the device behavior.

3.2.7 Controllability and Observability in digital systems

During Functional Verification, the way to direct the signals is a fundamental aspect. The control of the stimulus at the input of the device can increase the coverage and help to detect bugs. If a bad control of the signals is performed then less regions could be covered. Therefore, techniques which make a good control of the signals allow exercising and covering most of regions of the behavior. The controllability can be defined as follows:

Definition 13 *The "Controllability" can be defined as the ability of a testbench to generate stimulus which can exercise every part of the device behavior. Therefore, it is the ability to change the values in every node to obtain a determinate value at the device output.*

This means that not all stimuli are suitable to reduce the time and increase the advance of exercised behavior because of the signal management in the device under verification.

On the other hand, the change of state of different signals needs to be detected in the probe points. If an error in one or more signals is activated but is not propagated at a probe point, then, the error could be undetected. The observability is defined as follows:

Definition 14 *The "Observability" is the ability to shift an error to a place where it can be observed. This can reduce the cost of the error detection by the verification environment.*

Under this condition is important in order to detect the cause of the errors and improve the device implementation.

Based on the last concepts we can mention of “Testability” which is the difficulty observing and controlling the logical values of internal signals at the input and output of the digital system [57]. Figure 3.7 shows the observability and Controllability in the verification of a digital system.

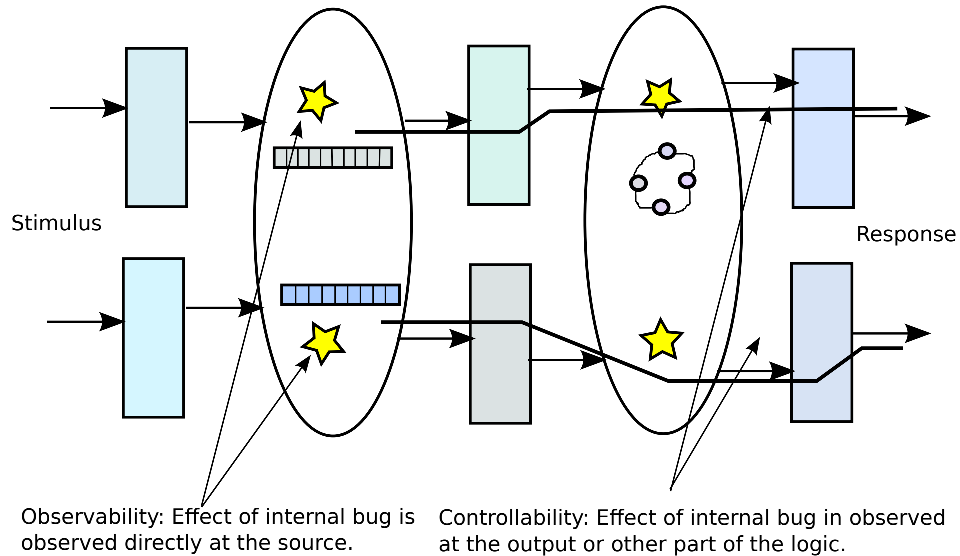


Figure 3.7. Observability and Controllability in a Digital system.

3.2.8 Pseudo-random Constraint Stimulus

Constraints define what and when the device stimulus need to be inserted at the device input. These are defined by the device behavior. A constraint can be represented as a Boolean formula over a design signals. There are two types of constraints: the environment constraints and the constraints which are used as test directives.

The simulation of constraint random stimulus can be made when the stimuli meet certain requirements of the environment and can be used to reach hard cases of the device behavior.

The feedback of the functional coverage can be used for the constrained random generation in order to direct the constrain solving and the randomization to exercise the not yet covered behavior.

Generating stimuli is required to fully exercise the device, in other words, to cause completely display all possible behaviors. Some techniques that have been used are:

- Generating random test program (RTGP).
- Based test generators models (MBTG).

In performing verification of a device we should take appropriate account ranges for stimulus. The stimulus generation can be composed of constants and generating sequences. Restrictions generation has rules, which control the generation of the input data. Data streams that are sent to the device to produce coordinated actions. The restrictions generation is divided into groups based on the source:

- The functional specification of the device: are those functional restrictions that are required to validate the device.
- The verification plan: are those that make a useful subset for verification.

Both types of restrictions are necessary to reduce the amount of all valid stimuli to those who exercise the necessary conditions limit the device. The specifications concerning the generation of stimuli must be referenced by the section of the verification plan.

Figure 3.8 shows the constraint random test generation process. The process begins with the pseudo-random generation of a set of tests. After that, the device simulation is performed, then the coverage analysis is made. Using the coverage information obtained if a set of holes is identified, then some minimal code modifications are implemented in order to generate a directed test case or a new pseudo-random test in generated using different seeds. The process is repeated again until a stop criteria is met.

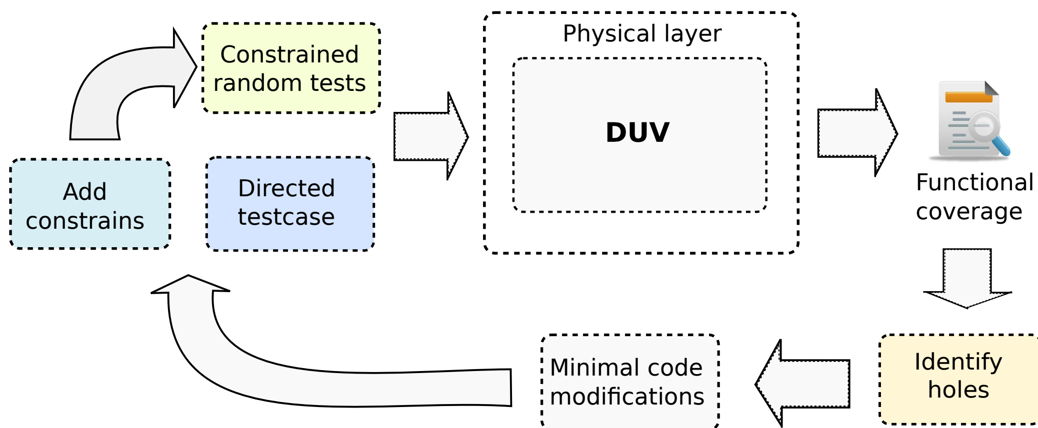


Figure 3.8. Constraint random test generation.

3.2.9 Modeling Functional Verification through Hardware Description Language (HDL)

When the engineers develop a hardware device the digital system needs to be modeled by means of a Hardware Description Language (VHDL, Verilog, SystemVerilog, SystemC). The original intention is a fundamental part to design a digital system because the design engineer translates the requirements from this intention to lines of code using the hardware description language and the structures from the programming language.

On the other hand, the verification engineers can model the functionality of the device and the input stimulus using different software tools through a simulation process. The simulation is the process which allows modeling the full behavior of a device. When the device is modeled by means of a software tool the real response speed of the semiconductor device is represented at a fraction. It means that the waveforms of signals are shown using a determined time according to that software tool.

Different environments can be modeled using a Hardware Description language such as: Verilog, SystemVerilog, VHDL, etc. The engineers model the wished environment according to the conditions which are needed to exercised the device behavior. Moreover, in order to check the correct behavior and detect the changes of the behavior different modules such as: Score boards, Monitors, among others are created.

Figure 3.9 shows the modeling process of a hardware design by means a Hardware Description Language. At the beginning, the design engineer reviews the original intention from the specification. After that; the functionality of the design is modeled in a black box level, then, this functionality is mapped to code lines based on a Hardware Description Language (HDL). When a implementation is made, this is tested using a simulation tool and the results obtained are analyzed.

3.2.10 Proving the Design through the Simulation

Emulation of digital systems is a process where the implementation is mapped to a physical device that can be a FPGA, logic arrays, etc. and is driven by means of a testbench or by a real environment where it is supposed that the device will be placed. The objective of emulation is to accelerate the simulation in hardware.

In the case where a testbench is used from the computer the latency of the communication with the device can be an order of magnitude slower than the emulation using only the real

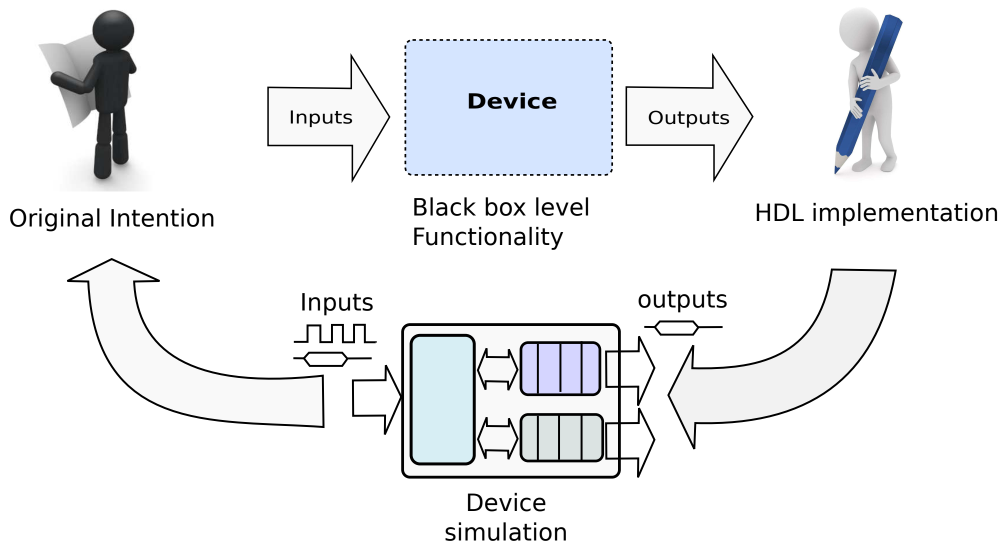


Figure 3.9. Modeling a device through HDL simulation.

environment.

Given an initial specification of a design, designers move these specifications to a hardware description language such as SystemVerilog, VHDL, Verilog or SystemC. This description is given to register transfer level (RTL), in which, the functionality between combinatorial blocks and different building blocks of a digital design is specified. RTL implementation is synthesized at a network of transistors or logic gates, called gate level description. From a design we can go through the behavioral, RTL, and gate levels. Given these different levels, it is important to verify the initial description and the equivalence of descriptions at different levels of abstraction involved in the design process.

Figure 3.10 shows a flowchart of verification of a digital system based on the simulation. The success of the check depends on how to analyze the results to conclude whether the digital system meets a proper operation.

Formal verification techniques have shown to be effective for tasks such as verifying the equivalence of two circuits at different levels of abstraction. However, at simulation level it remains a critical part of the verification process. It is necessary to order the test vectors to simulate a design, and lookup tables should be appropriate to exercise all aspects of the design. Later, when the design is made, the test vectors of the input-output can be applied to the design through to detect systematic failures caused in the manufacturing process tests. Test vectors must be available at each level of abstraction of the design to ensure that it is possible to verify

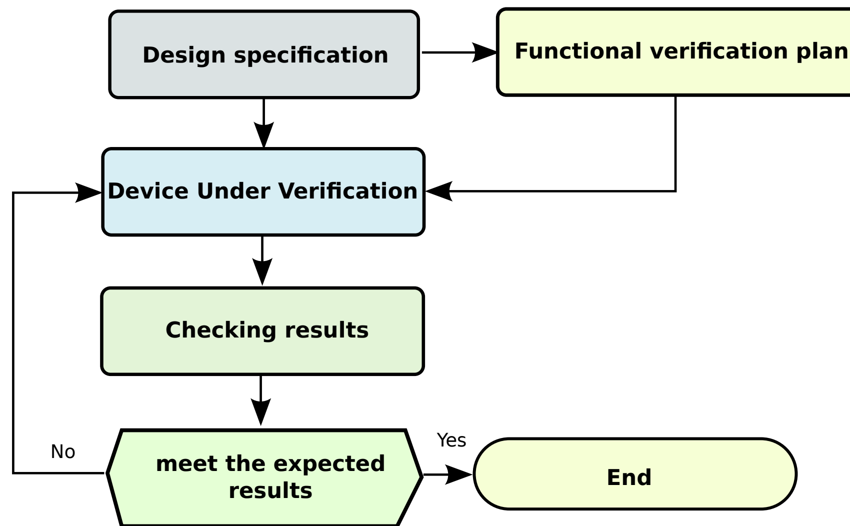


Figure 3.10. Flowchart of verification based on simulation.

the design.

The methodology based on simulation consists in simulating the design for all vectors in a set of tests (*test*) within an environment which models the current system hardware, and the simulation outputs are reviewed to analyze the behavior and review if the system meets specifications. Test vectors can be obtained through generators or pseudo-random vectors and they can be entered by designers based on the functional specification of the design. With these methods it is difficult to ensure a degree to which the design has been proven.

Figure 3.11 shows the main HVL techniques used to perform the verification process. The general method based on simulation involves the application of techniques for generating test according to the level. After each level of the design is completed the vector sequences are generated and then the device is simulated at that level of abstraction and lower levels. The results are compared through different levels to check if they match. However, the test generation process is complicated and time consuming. There are two problems associated with validation through simulation, which are: coverage and generation of simulation inputs.

3.3 Resume

In this chapter the main concepts about the functional verification of digital systems were described. Functional Verification includes a set of steps that involves a general methodology to

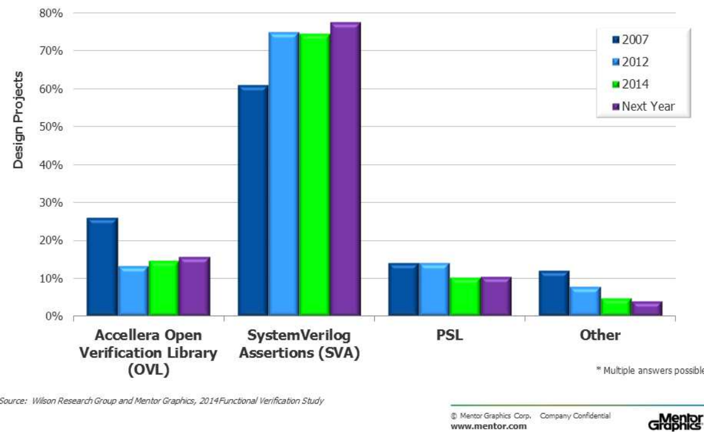


Figure 3.11. Trends in Functional Verification Techniques.

be followed today. In addition to measuring the progress of the functional coverage of digital systems, it is necessary to use a coverage metric or measure that allows to know the progress of the coverage advance during the test of a device. Among the most commonly coverage metrics are: paths coverage, statement coverage, coverage of state machines, fault coverage.

A definition of functional coverage was also described, which can be understood as the percentage of the device behavior that has been verified. We may also mention the importance of using a good coverage model for proper verification of a device, as this represents the device behavior and thus a space of inputs and outputs that are mapped into the model. Also, the concepts about digital design, modeling and simulating were described. These concepts are important to understand the principles of functional verification and the current challenges generated from the technology advances.

In short, these concepts will be useful to understand the proposed method in chapter 5. In the next chapter the concepts of binary meta-heuristic algorithms are presented. Moreover, their principles, differences and advantages with respect to other types of algorithms are described.

CHAPTER IV

Meta-Heuristics on the Binary Space

Chapter 4

Meta-Heuristics on the Binary Space

The word heuristic is derived from the verb $\varepsilon\nu\rho\iota\sigma\kappa\varepsilon\iota\nu$ which means to find, to invent or to discover. The Greek word *meta* means beyond or upper level. According to Fred Glover in his seminal paper [58] a meta-heuristic is a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimization. Also, the heuristics can be seen as a criteria, methods or principles which decide between several possible options to be the most effective in order to achieve some goal.

Meta-heuristics can produce a reduction of possible solutions in the search space. This is because they use different strategies to explore and exploit the search space. Initially different points in the search space are explored using pseudo-random generated data. After new points are chosen based on the interaction and the fitness values of the points previously evaluated. These points represent possible solutions in the search space. By finding points near a global solution, the areas near these points are largely exploited, and what we see depends on the convergence in which these algorithms can find the area that contains a solution with a higher fitness value compared with others in that space without being trapped in a relatively good but not necessarily in areas where the best solutions are.

In the present chapter, the principles of the Genetic algorithm, Particle Swarm Optimization algorithm, Differential Evolution algorithm, Compact Genetic algorithm are presented. Also, the characteristics of the meta-heuristics on the binary spaces are described.

4.1 Binary search space

Different heuristic methods have been proposed to solve optimization problems in continuous spaces. However, there are problems which are represented over discrete search spaces; one of such spaces is the problem of Functional Verification of digital systems because the binary representation is implicit into the discrete domain of digital systems.

Unlike others, the binary representation can be used to represent a wide range of characteristics, numbers, variables, etc. For example, the binary number: 101011 may represent the number 43 in decimal but can also represent the presence of four features or the absent of 2 in a particular set of variables.

Adding a bit to a binary string the search space is doubled. This can be exemplified by initiating a single binary digit, which can represent a point in the binary space that can take the values zero or one. Adding a bit we can have four different combinations: $\{(0,0),(0,1),(1,0),(1,1)\}$ the set of points could be represented by a square in two dimensions, where each vertex contains a possible tuple of binary values and each edge represents the distance between each tuple. For example, with the first point $(0,0)$ neighboring corners will be $(0,1)$ and $(1,0)$ the opposite corner will be $(1,1)$. Adding a bit more, three-digit binary strings have 8 possible values $(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)$. This set of 3-tuples could represent a cube where each vertex has a possible value of 3-tuples. Increasing the binary string with more digits, this gives an almost incomprehensible increase in the search space, as each new binary digit adds another dimension to all previous points (This is known as a hyper-cube). For example, if we have a 20-bit binary string, the search space S is equivalent to $S = 2^{20} = 1,048,576$. This complicates the search for a point within the set of all possible options. That means trying to find a point which meets the solution to a problem in this space may be intractable. Even in real problems which require finding good solutions in a reasonable amount of time, it is impractical to perform an exhaustive search of all possible points.

Because of the issues described, it is necessary to use algorithms that make a search in this binary space and find the optimal points within a reasonable time. At the same time, this process is based on the information obtained to evaluate possible solutions through directed functional verification. In the next section the main aspects about evolutionary algorithms are presented.

4.2 Evolutionary algorithms

Most times meta-heuristic provides a good solution in a modest time period. Meta-heuristics involves the evolutionary algorithms. According to A. E. Eiben and J. E. Smith [59] these algorithms are based on the evolution theory where given a population of individuals within some environment that has limited resources, competition for those resources causes natural selection (survival of the fittest). This in turn causes a rise in the fitness of the population. The selection of better individuals produces a better solution. There are two fundamental operators that form the basis of evolutionary systems:

- Variation operators (recombination and mutation) create the diversity of the population which permits performing a good search in the search space.
- Selection permits making a difference between the quality of the solutions.

Recombination is an operator which is applied to two or more candidate solutions (called parents) producing one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Then, the application of these operators creates a new population (offspring). The process can be iterated until a termination condition is met or whether the best solution is reached by the algorithm.

The genetic algorithms have been used to make the search for the functional verification of digital systems. In the works which were described in the state-of-the-art the researches used the basic genetic algorithm to search for vector sequences optimizing the values of coverage in the test of digital systems. However, other meta-heuristics have been proposed among which are: Particle Swarm Optimization algorithm (PSO), Differential Evolution algorithm (DE), Compact Genetic algorithm, among others. In this chapter the binary versions of these meta-heuristics will be described.

Different from other types of algorithms the meta-heuristics represent mechanics which reach good solutions in the search space using a reasonable amount of time. These algorithms take decisions based on the evaluation of the possible solutions. The change in the algorithms performance can be seen more clearly when the problem implies a big search space and the use of exhaustive search is not practically. In real problems, the time used in order to evaluate the digital systems is very expensive. Even for devices with a regular size it is not practically to try all possible options because the regions of interest to be covered into the behavior space requires to use good strategies in order to reach sequences in a suitable time. This is the reason why the use of meta-heuristic algorithms is the main proposal of this research.

Another important aspect is the concept of optimization. In this research the term global optimization is defined as the process of attempting to find the solution x^* out of a set of possible solutions S that has the optimal value for some fitness function f. In other words, we are trying to find the solution x^* such that $x \neq x^* \Rightarrow f(x^*) \geq f(x)$. In this case, a maximization problem is assumed. These meta-heuristic algorithms have the ability to maintain a diverse set of points which provides not only a means of escaping from local optima, but also a means of coping with large and discontinuous spaces. In the next subsection the main characteristics of the genetic algorithm will be described.

4.2.1 Binary Genetic Algorithm

The genetic algorithm was proposed by John H. Holland in the early 1960's was motivated to solve machine learning problems. This algorithm was designed based on the Darwinian theory where the interactions between the individuals of the population can produce better solutions. After different epochs better solutions are found.

The genetic algorithm uses binary strings (genotypes) to represent each possible solution (phenotypes). Every binary string is composed of a set of positions (genes), a value into each position (allele) and a set of operators (crossover, mutation and selection). The genetic algorithm uses the principle based on population of individuals where they share the characteristics in order to produce better individuals and reach a good solution of the problem. There are five basic components in order to apply the genetic algorithm:

- A representation of potential solutions of the problem.
- A way to create the initial population (commonly a pseudo-aleatory process).
- A cost function which plays the role of environment, classifying solutions by mean their aptitude value.
- Genetic operators to modify the composition of the individuals to be produced in the next epoch.
- Values of the parameters (population size, crossover probability, mutation probability, number of epochs, etc.).

Algorithm 1 shows the pseudo-code of the binary genetic algorithm. There are four main steps which are executed in every epoch until a stop condition is not meet. The binary genetic algorithm uses operators of crossover, mutation and selection in order to create a new population.

Different parameters need to be configured: crossover percentage, mutation, selection in order to perform the search.

Algorithm 1: Binary Genetic algorithm

while *Stop condition is not met* **do**
  Generate a initial population.
  Calculate the fitness values for every individual.
  Select (probabilistic) based on aptitude.
  Apply genetic operators (crossover and mutation) in order to generate the offspring.
end

The crossover operator is the recombination of two or more Individuals (parents), which are selected using different criteria based on their value aptitude. Generally, the operation of crossing is done by taking a reference point within the binary string and combining each of the sub-strings of the two parents to produce two children. This operation can be repeated with other individuals to form a new population for the next generation.

The mutation operator or alteration consists of small changes in some positions of binary strings. Generally, these changes are made in a pseudo-random way and produce a new solution. This new solution must be evaluated and will constitute part of the new population.

Other step of the genetic algorithms is the selection of individuals. This consists of selecting two or more individuals in order to apply the crossover and mutation operators. There are different criteria to select the individuals. After new population is created then a new iteration is performed and the stop criteria is checked. In the next subsection, the Particle Swarm Optimization (PSO) algorithm will be described in its binary version.

4.2.2 Binary Particle Swarm Optimization algorithm (BinPSO)

In 1995, Eberhart and Russel James Kennedy [60] proposed a new algorithm based on particle swarm intelligence. The algorithm was based on the psychological and social theory, which suggests that individuals moving through a social-cognitive space should be influenced by their previous behavior and the better performance within their neighborhood. This influence can be the better of all population or only near to the particle.

In this algorithm, each individual of the population is named as a particle and these particles are grouped into neighborhoods, which constitute the population. Each particle has a social and a cognitive behavior. There are two types of configurations: *gbest* and *lbest*. The first configuration can be interpreted as *gbest* which connects a particle with all others in the population. This causes the behavior of each individual to be influenced by the best performance of any particle in the population. The second configuration *lbest* creates a set of swarms consisting of a number of particles. The behavior of each particle is influenced by the best performance of a particle within the swarm.

Algorithm 2 presents the pseudo-code of the Particle Swarm Optimization algorithm with binary representation proposed by James Kennedy. The first step is to initialize every value of particles. The fitness value of each is then calculated, then, the value obtained with the value of their previous best position is compared. If best previous position is replaced by the current, then, the values of velocity are calculated and then compared. If the best position is less than the value calculated of the sigmoidal speed function, then x_{id} is set to 1 but takes the value of 0. The process is repeated again until a top condition is satisfied.

The probability that each individual decides whether an element is 1 or 0, is defined as a function of personal and social factors. This function is defined in Eq. 4.1.

$$v_t(t) = v_{id}\,(t-1) + r1 \times \varphi_1\,(p_{id} - x_{id}\,(t-1)) + r2 \times \varphi_2\,(p_{id} - x_{id}\,(t-1)) \tag{4.1}$$

Where:

- $v_{id}\,(t-1)$ is a measure of the individuals predisposition or current probability of deciding 1.
- φ_1 is a positive random number drawn from a uniform distribution with a predefined upper limit.
- $r1$ and $r2$ are positive random numbers between 0 and 1.
- $x_{id}\,(t)$ is the current state of the bitstring site d of individual i.

Algorithm 2: The Binary Particle Swarm Optimization algorithm (BinPSO)

```
swarm ← initSwarm()
bestParticlePosition ← getBestParticlePosition()
globalBest ← getGlobalBest()
while / terminationConditionMeet do
    while / terminationConditionMeet do
        Loop
        for i = 1 → number of individuals do
            Compute:
            if G(x_i) > G(p_i) then
                for d = 1 → dimensions do
                    p_id = x_id
                    next d
                end
            end
            g = i
            for j = indexes of neighbors do
                if G(p_i) > G(p_2) then
                    g = j
                end
                Next j
                for d = 1 → number of dimensions do
                    v_t(t) = v_id(t − 1) + r1 × φ_1(p_id − x_id(t − 1)) + r2 × φ_2(p_id − x_id(t − 1))
                    v_id ∈ (−V_max + V_max)
                    if ρ_id < s(v_id(t)) then
                        x_id(t) = 1;
                    else
                        x_id(t) = 0;
                    end
                    Next d
                    Next i
                    Until criterion
                end
            end
        end
    end
    swarm ← restartSwarm(swarm, globalBest)
end
```

- t means the current time step, and $t-1$ is the previous step.
- p_{id} is the best state found so far, for example, it is 1 if the individuals best success occurred when x_{id} was 1 and 0 if it was 0.
- p_{gd} is the best neighborhood, again 1 if the best success attained by any number of the neighborhood was when it was in the 1 state and 0 otherwise.

Since the original version of the Particle Swarm Optimization algorithm was designed to work in the domain of real values, so a function was proposed to change the values of each element of the particles using a probability value. The Sigmoid function in equation 4.2 is used to change the value 1 or 0 of each bit that is part of the particle, depending on the probability threshold. If the value v_{id} is higher, the particle is more likely to select 1 and if the value is

lower, it is more likely to select a 0. The v_{id} value is restricted in the range [0.0,1.0].

$$S\left(V_{id}\right) = \frac{1}{1 + exp\left(-v_{id}\right)} \tag{4.2}$$

How to control the influence of the traveled path by each particle and the influence of the other particles in the population may cause the particles move quickly to regions with better fitness values. To do this, pseudo-random values are used, which produce similar effects to the mutation operator in genetic algorithms. In the next subsection the main characteristics of the binary differential evolution algorithm are described.

4.2.3 Binary Differential Evolution Algorithm (DE Algorithm)

Differential Evolution algorithm was proposed by Rainer Storn and Kenneth Price [19]. This is an evolutive algorithm based on the difference between two individuals and a crossover mechanism to generate new possible solutions. One advantage of the algorithm is that only best solutions are used to create a new population. The algorithm has obtained good results in many optimization problems [20]. Differential Evolution algorithm have been successfully in different optimization problems [18]. The Differential Evolution algorithm has shown good results in problems with discrete and real spaces.

Although, the initial version of the Differential Evolution algorithm was proposed for problems with real representation of variables, obtaining better results in most cases [61]. Later, different works were carried out for binary representation [62, 63]. Application of the original version of the Different Evolution algorithm in discrete spaces is complicated because of the origin of the operators which are used in the original version of the Differential Evolution algorithm. Due to this, it is necessary to use operators to let the algorithm apply over discrete spaces. Different works were carried out using new crossover and mutation functions in discrete spaces [64–68].

In 2007, A. P. Engelbrecht and G. Pamparà published a paper [64] which describes different strategies by the Differential Evolution algorithm. The obtained results show good solutions in different optimization problems. On the other hand, in the work [65] in 2011, Changshou Deng, Bingyan Zhao, et al. proposed a version of the algorithm based on a mutation function which can be used in the discrete domain. The mutation function is defined in the equation 4.3.

$$h_{ij}(t+1) = MOD_2(x_{r3j} + (x_{r1j} \;\; XOR \;\; x_{r2,j}) \tag{4.3}$$

Where:

- MOD_2: is the function of addition module 2.
- $h_{ij}(t+1)$: is the is the offspring generated.
- x_{r1j}: is the individual selected by a pseudo-random number r_1.
- x_{r2j}: is the individual selected by a pseudo-random number r_2.
- x_{r3j}: is the individual selected by a pseudo-random number r_3.
- Based on these experimental results we selected this mutation function.

Algorithm 3 represents the pseudo-code of the algorithm of Binary Differential Evolution which was proposed by Rainer Storn and Kenneth Price with a crossover and mutation function proposed by Changshou Deng, Bingyan Zhao, et al. [65]. At first, an initial population is generated pseudo-randomly. Each of the vectors is evaluated in order to obtain a fitness value. After, three different vectors are selected from the population. Once three vectors are selected, based on a predefined percentage value CR the operators of crossover and mutation are performed. These are based on the difference of two of them, which is added to the third selected. If a random value is higher than CR value then, the corresponding element of the best current vector is taken. If the new vector is better than the current, then the new vector replaces the original vector in the population.

Algorithm 3: Binary Differential Evolution Algorithm

G = 0
To generate the initial population: $\vec{x}_{i,G} \forall \ i \ \ i = 1, ...NP$
To evaluate every vector: $f(\vec{x}_{i,G}) \forall \ i \ \ i = 1, ...NP$
Calculate: **for** $G = 1 \rightarrow MAX_GEN$ **do**
    for $i = 1 \rightarrow NP$ **do**
        $Select \ \ randomly \ \ r_1 \neq r_2 \neq r_3 : j_{rand} = randint(1, D)$ **for** $j = 1 \rightarrow D$ **do**
            if $rand_j[0,1) < CR \ \ or \ \ j = j_{rand}$ **then**
                $U_{i,j,G+1} = MOD_2(x_{r3,j,G} + (x_{r1,j,G} \ \ XOR \ \ x_{r2,j,G}))$
            else
                $u_{i,j,G+1} = x_{i,j,G}$
            end
        end
        if $f(\vec{u}_{i,j,G+1} \leq \vec{u}_{i,j,G})$ **then**
            $x_{i,j,G+1} = u_{i,G+1}$
        else
            $x_{i,j,G+1} = x_{i,G}$
        end
    end
    G = G+1
end

Figure 4.1 shows the generation of new vectors based on the addition of the difference between two vectors and a third vector which were chosen randomly. In the case of the problems with binary domain that difference between vectors can be mapped in a binary difference.

Therefore, the operations between bits generate new vectors $U_{i's}$ which are evaluated and compared with the originals in order to choose the vectors to keep and will be replaced in the new population. In the next section the main characteristics of Compact meta-heuristic algorithms

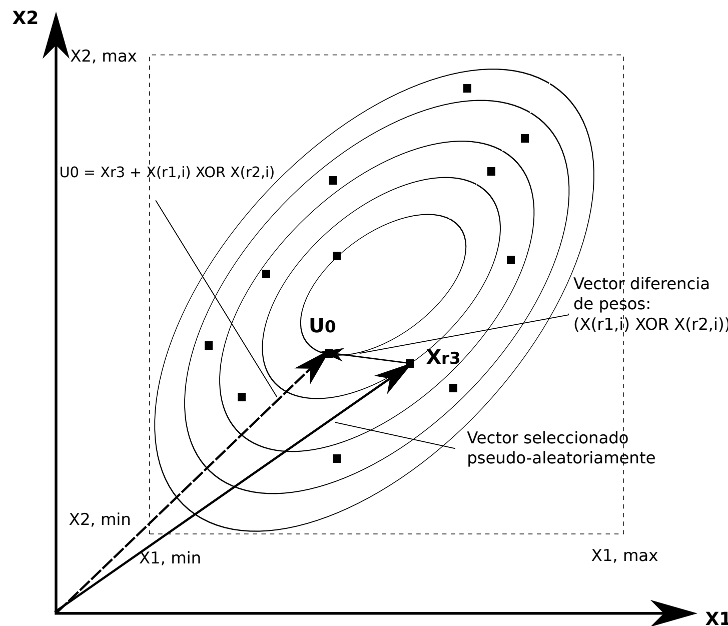


Figure 4.1. Generation of new vectors in the Binary Differential Evolution algorithm.

will be presented.

4.3 Compact meta-heuristic algorithms

The compact meta-heuristic algorithms are reduced versions of the meta-heuristics. Because these techniques use a reduced number of individuals in the population, there is a reduction in memory requirements in hardware devices. One of the strategies used to reduce memory requirements consists of representing the population as a probability distribution over the set of possible solutions. Unlike the meta-heuristics based on population, the Compact meta-heuristics are based on the increase of the probability distribution by each bit position of the possible solutions.

Different works have been carried out, i.e., a compact genetic algorithm (cGA) which was proposed by Fernando G. Lobo, Georges R. Harik and David E. Goldberg in [69]. In

this work, the authors proposed a genetic algorithm which reduced the memory requirements using a probability distribution for the population. The strategy consisted in obtaining different probability values for each individual bit based on the fitness function. Only one vector represented the probability distribution of the population to generate new individuals and find a better solution.

Another work is [70] where Chang Wook Ahn and R. S. Ramakrishna presented two new versions of the original compact genetic algorithm using the elitism concept (pe-cGA and ne-cGA). The elitism consisted in saving the best current solution to keep it in the new iteration. In this case, they presented persistent elitism and non persistent elitism concepts. In the first case, the best solution is saved in order to be compared with a new possible solution in the next generation. In the second case, the best solution is saved only if a value criteria maintains a determinate length of inherent. According the results using a set of optimization problems, these versions improved the original version of the compact genetic algorithm proposed in [69].

Recently, a compact version of the Differential Evolution algorithm was proposed by Francesco Cupertino Ernesto Mininno, Ferrante Neri and David Naso in [71] for problems with real representation of variables. In this work, they used the principle of the Compact genetic algorithm because the algorithm stored a reduced number of individuals as a population by means of statistic values. The algorithm was applied to solve different numeric problems, obtaining very competitive results. Also, to improve the performance of the algorithm, some work carried out with a local search has been proposed. In the work [72], Ferrante Neri and Ernesto Mininno carried out a Compact Differential Evolution algorithm with a simple Local Search mechanism for real representation. The algorithm was applied to solve a control system design problem for a Cartesian robot for variable mass movements, and it obtained good results.

Unlike other works, in this research the proposed Compact-BinDE algorithm is used to maximize the functional coverage percentage in the verification of digital systems in the binary domain. The algorithm follows the principle of these compact meta-heuristic algorithms based on the probability values over the set of possible solutions, and the binary mutation and crossover of the Differential Evolution algorithm with a simple local search. In the next subsection we explain the principle of the compact genetic algorithm.

4.3.1 Compact Binary Genetic algorithm

The Compact Binary Genetic algorithm was proposed by Fernando G. Lobo, Georges R. Harik and David E. Goldberg in [69]. The original version was based on the principle of random walk model proposed by George Harik [73]. The algorithm uses a probability distribution over the population. These probability values are saved in a vector which represents the information for every bit position of the individuals from the population.

Different from the original Binary Genetic Algorithm the compact version can save memory resources because it saves only the proportion of ones and zeros generated by mean evaluations and, the original Genetic Algorithm needs to store n bits for each bit position. Therefore, the Compact Binary Genetic Algorithm does not replace the original Binary Genetic algorithm but it saves the memory requirements. This characteristic is useful specially in problems where a huge population size is required.

Selection of the individuals is based on samples from the PV vector in order to produce a new individual and save the information after to evaluate how good is that solution. The crossover is performed based on the added information in the PV vector and the selection of every element from this vector.

Algorithm 4 represents the pseudo-code of the Compact Genetic algorithm using non elitism. The algorithm can be described as follows: at the beginning, the p vector is initialized with 0.5 in all bit positions. The population size n and the chromosome length are configured according the values required. After that, every iteration a and b values are sampled from p vector. Then, a competition is made between these values and the winner and loser are assigned with these values respectively. After that, every bit position is compared using the values of winner and loser the values of p vector are increased or decreased by $1/n$.

A compact version of Genetic algorithm with Elitism was proposed by Chang Wook Ahn and R. S. Ramakrishna in [70]. This version uses the Elitist individual which is a survivor in the next generation. Different from the original version of the Compact Genetic algorithm, the Elitism version takes the best individual in a persistent way. This means that in every iteration the best solution is kept. Two different versions were proposed: persistent Elitism and non persistent Elitism. The non persistent Elitism is based on a parameter η that is a specified length which is used to determine when the Elitism individual is kept for the next iteration or if it is replaced by a new individual. According the obtained results the algorithms outperform the original version of the Compact Genetic algorithm.

Algorithm 4: Pseudo-code of the Compact Genetic algorithm

```
n: population size, l: chromosome length;
Initialize probability vector;
for i = 1 → l do
 | p[i] = 0.5
end
Generate two chromosomes from the probability vector;
a = generate(p);
b= generate(p);
Let them compete;
winner, loser = compete(a,b);
Update the probability vector;
for i = 1 → l do
 | if winner[i] ≠ loser[i] then
 |  | if winner[i] == 1 then
 |  |  | p[i] = p[i] + 1/n
 |  | else
 |  |  | p[i] = p[i] − 1/n
 |  | end
 | end
end
Check if the probability vector has converged.
Go to step 2, if it is not satisfied.
The probability vector represents the final solution.
```

Algorithm 5 shows the pseudo-code of the steps for persistent elitism of the compact binary genetic algorithm. In the algorithm, the best solution is saved in the next generation and is compared with the solution of the next iteration. If the new solution is better than the best solution then this replaces the best solution. According to Chang Wook Ahn and R. S. Ramakrishna this version produces better results than the original version of the compact genetic algorithm.

Different from the versions of meta-heuristics based on population the compact version of genetic algorithm reduces the memory resources because it uses the statistics values of the population by means of a vector of probabilities. This allows the application of these algorithms in hardware devices with low memory resources.

Algorithm 5: Modification of the cGA with pe-cGA.

E_{chrom}: elite chromosome, N_{chrom}: new chromosome.
Step 2: Generate one chromosome from the probability vector.
if *the first generation* **then**
  $E_{chrom} = generate(p)$ /* initialize the elite chromosome */.
$N_{chrom} = generate(p)$ /* generate a new chromosome */.
Step 3: Let them compete and let the winner inherit persistently.
winner, loser = compete(E_{chrom}, N_{chrom});
$E_{chrom} = winner$ /* update the elite chromosome */

4.4 Resume

In this chapter some concepts about binary meta-heuristic algorithms were presented. The Binary Genetic, the Particle Swarm Optimization and the Differential Evolution algorithms were described. Different from the problems with variables of real representation, the problems involving a discrete domain require the use of meta-heuristics using operators which permit making an operation between variables of binary representation. There are different meta-heuristic algorithms which perform the binary search using different philosophies. Such algorithms have been successfully used in different optimization problems and almost have been adapted and implemented as part of the test vector generation for functional verification devices.

Also, a version of a Compact genetic algorithm was described. Different from meta-heuristics based on population this type of Compact meta-heuristics use a probability vector to represent the population over the set of statistic values. This strategy reduces memory requirements to implement the algorithm over a hardware device. This principle is used in the proposed binary differential evolution algorithm. In short, these concepts will be useful to understand and propose the methodology presented in the next chapter.

Finally, meta-heuristics are methods that have been used for problem solving where it is difficult to obtain a good solution in a reasonable time, and have an efficient performing of search based on findings obtained in the evaluation of possible solutions. These characteristics and given the previous work of the state-of-the-art allows us to propose the use of meta-heuristics to optimize coverage values making a proper search in the discrete space.

In the next chapter the proposed test generation method will be presented. Besides, the meta-heuristics used, coverage models, devices as well as the construction of the platform used for the functional verification process is given.

CHAPTER V

Proposed Method

Chapter 5

Proposed Method

Search for test vector sequences in all binary space of digital systems is very expensive because the extensive search is inefficient and it requires much time to evaluate the device under verification. Moreover, long binary sequences represent a big problem when different blocks of bits are required in order to test the functionality. Therefore, to try to search appropriated test sequences which exercise specific points is not a trivial problem; different CoverPoints can be covered excessively representing a waste of time for the simulation and the points which were not covered represent holes. These points could require special analysis because can represent regions of the behavior which were not implemented during the design step.

In this research we propose the use of meta-heuristic algorithms in order to maximize the percentage of coverage points during the functional verification, reducing the number of holes at the same time. Also, the functional coverage models map the functionality of the device to the CoverPoints, therefore, they represent the full behavior of the device under verification.

The method includes a binary Particle Swarm Optimization (PSO) algorithm and a compact binary differential evolution (Compact-BinDE) algorithm [74]. The test generation method directs the search using the coverage information obtained at the end of the simulation. Also, the proposed software platform is used in order to evaluate the device and control the interaction between the DUV and the verification environment. Different from other methods based on meta-heuristics, the proposed method uses functional coverage models, proposed fitness functions and these meta-heuristics in their binary versions. The use of these versions is because of the translation between the phenotype (test sequences) of the solutions in the verification problem and genotype (bit-strings) is implicit. This genotype is used by these algorithms in order to generate test vector sequences which cover the set of CoverPoints.

Other important aspect of this thesis proposal is the development of a proposed soft platform in order to communicate the device simulation tool with the algorithms. To do this, an interface between the hardware description languages (verilog, systemverilog) and C language was developed. Different modules to analyze, save and calculate the fitness functions were implemented. An objective to do that was the automation of the functional verification process.

In the present chapter the proposed method is described. Moreover, the use of the meta-heuristic algorithms and the characteristics of the software platform are presented.

5.1 Proposed Compact Binary Differential Evolution Algorithm (Compact-BinDE)

In this research we propose a version of a Compact binary Differential Evolution algorithm (Compact-BinDE) to maximize the functional coverage percentage in the functional verification process. Before describing the characteristics of the Compact-BinDE algorithm, we need to understand the relationship between the meta-heuristics and the functional verification. To begin, the directed functional coverage generation can be interpreted as a heuristic which searches a set of binary sequences to test the device behavior using the coverage information obtained at the end of the device simulation. Also, the set of sequences needs to reach a high functional coverage percentage. In other words, in every iteration we expect to obtain a set of test sequences which covers an increasing number of functional characteristics of the Device Under Verification (DUV).

However, at the beginning, we do not yet know which set of sequences is the best, and the number of possible options is increased proportionally to the complexity of the functionality of the device. Due to this, it is necessary to use a technique which can search the best options by doing an efficient binary search.

Also, the principle of the Differential Evolution algorithm is based on the selection of three test sequences done in a pseudo-random way. One of them is chosen as a base vector and the difference between the other two binary sequences is added to this. This difference between the two binary sequences represents the new direction and the step which will be taken by the first vector to produce a new possible solution. The new direction directs the new vector towards a better solution. At the beginning, the difference between two sequences is long because one sequence is distant from the other. When more information is obtained from the simulations,

the difference is reduced and better solutions are found. In every iteration the new test sequence is evaluated and compared with the best current solution to decide which is the best. The best solution will be replaced by a new sequence only if the new sequence has an equal or better fitness function value.

Based on these ideas, we propose a Compact Binary Differential Evolution algorithm (Compact-binDE) using the Differential Evolution algorithm and the compact meta-heuristics principle. Moreover, in the proposed algorithm, a simple local binary search is added. The proposed version is based on the compact version of the genetic algorithm and the compact Differential Evolution algorithm described in the last chapter, using a mutation mechanism similar to the work [65] for binary representation. Also, a local search based on a pseudo-random changes of the best solution is added to improve the search during the evaluations of the circuit. The local binary search modifies the best binary solution in order to try to improve the solution, this is made through different probability values [72, 75]. Other techniques could also be applied to make the local search.

It is important to mention the concept of mutation. In general, a mutation is defined as an alteration or change. However, in Evolutive algorithms, a mutation is seen as a change with a random element or random value. This means that a change will be produced using a probability distribution function (PDF). In this case the Differential Evolution algorithm uses a uniform distribution. The algorithm 6 shows the pseudo-code of the Compact Binary Differential Evolution algorithm (Compact-BinDE) with binary Local Search.

In the algorithm 6 PV represents the probability vector for each one bit N position of the sequence. P_{ls} value is used to apply the mutation and crossover of BinDE or the local search for each individual. $\vec{elite}$ Is the best current sequence. CR is the value of the crossover probability for each bit of the sequences. MAX_GEN is the maximum number of generations for the algorithm.

The algorithm is described as follows: First, each element of the probability vector PV is initialized to 0.5 and the $\vec{elite}$ test sequence is generated by means of these values. This sequence represents the best current solution. After that, if a pseudo-random value rand[0,1] is greater than the P_{ls} value, then, the Mutation and crossover operations of the Differential Evolution algorithm are performed. In this case, three sequences: r0, r1, r2 are generated from the PV vector. Then, for each bit of length, the CR value determines if the mutation operation is applied or if the element of the new solution will be replaced by the element of $\vec{elite}$. The mutation mechanism consists of an XOR operation between r1 and r2. This binary difference

Algorithm 6: BinDE/rand/1/bin version of Compact Binary Differential Evolution algorithm (Compact-BinDE) with Local search.

```
G = 0
PV initialization PV_{i,G} = 0.5 ∀ i i = 1, ...N
Generate elite⃗ by mean PV vector.
Evaluate the fitness function f(elite⃗_{i,G})
Calculate: for G = 1 → MAX_GEN do
    if rand_j[0,1) > P_ls then
        Generate three vectors r⃗_i by mean of PV vector
        Generate a number pseudo-randomly j_rand = randint(1, D)
        for j = 1 → D do
            if rand_j[0,1) < CR or j == j_rand then
                X⃗'_{j,G+1} = MOD_2(x_{r0,j,G} + (x_{r1,j,G}  XOR  x_{r2,j,G}))
            else
                X⃗'_{i,G+1} = elite⃗_{i,G}
            end
        end
    else
        Local search is applied
    end
    if f(X⃗'_{j,G+1}) ≥ f(elite⃗_{j,G}) then
        elite_{j,G+1} = X'_{j,G+1}
    end
    Update of VP vector:
    for j = 1 → D do
        if winner[j] ≠ looser[j] then
            if winner[j] == 1 then
                PV[j] = PV[j] + 1/N_p
            else
                PV[j] = PV[j] − 1/N_p
            end
        end
    end
    G = G+1
    for j = 1 → D do
        if PV[j] > 0 ∧ PV[j] < 1 then
            exit = return
        end
    end
end
```

is added to the r0 vector and the MOD_2 ensure that the result remains in the binary domain. Then, the new sequence is compared with the $\vec{elite}$ to determine which is better. The winner solution represents the sequence with the best fitness value and the looser solution represents the sequence with the least fitness value. After that, the PV vector is updated according to the comparison between the elements of the winner and looser solutions. The comparison is performed as follows: For every element, if the element of the winner solution is different from the looser solution and this element is 1, then the value of PV_j is increased by $1/N$. If the element is zero, PV_j is decreased by $1/N$. When any element of the PV vector is over 0 and

less than 1, then a new iteration is performed. The stop criteria can be a specified number of evaluations or a given coverage percentage.

There are different ways to perform the crossover and the mutation for the Differential Evolution algorithm. The equation (5.1) shows the basic way of the algorithm known as: DE/rand/1/bin, the term "rand" (as in "random") is used because the base vector is chosen in a pseudo-random way. The value "1" is because only one difference between two vectors is added and the name "bin" is used because it is a binomial distribution. The XOR operator calculates the binary difference between two sequences and the MOD_2 operator maps the sum of that difference with the chosen vector in the binary domain. According to the same criteria, the configurations: DE/Best/1/bin, DE/Rand/2/bin, DE/Best/2/bin can be used as mutation operations of the Binary Differential Evolution algorithm.

DE/Rand/1/bin:

$$\vec{X'}_{j,G+1} = MOD_2(x_{r0,j,G} + (x_{r1,j,G} \;\; XOR \;\; x_{r2,j,G})) \tag{5.1}$$

DE/Best/1/bin:

$$\vec{X'}_{j,G+1} = MOD_2(x_{Best,j,G} + (x_{r1,j,G} \;\; XOR \;\; x_{r2,j,G})) \tag{5.2}$$

DE/Rand/2/bin:

$$\begin{aligned} \vec{X'}_{j,G+1} = MOD_2(x_{r0,j,G} + (x_{r1,j,G} \;\; XOR \\ x_{r2,j,G}) + (x_{r3,j,G} \;\; XOR \;\; x_{r4,j,G})) \end{aligned} \tag{5.3}$$

DE/Best/2/bin:

$$\begin{aligned} \vec{X'}_{j,G+1} = MOD_2(x_{Best,j,G} + (x_{r1,j,G} \;\; XOR \\ x_{r2,j,G}) + (x_{r3,j,G} \;\; XOR \;\; x_{r4,j,G})) \end{aligned} \tag{5.4}$$

Each configuration can be implemented in the algorithm, the results of which are described in section 6.3. In the next subsection we describe the proposed test vector generation method.

5.2 Proposed test vector generation method

The test vector generation method used in this research involves the necessary steps to perform the simulation of the device using the input sequences. To do this, the method generates sets of sequences through the use of the Compact Binary Differential Evolution algorithm (Compact-BinDE) and Particle Swarm Optimization (PSO) algorithm. The algorithms use fitness functions which are based on the number of holes and covered points obtained during the run time simulation. This method focuses on the coverage points that are not completely covered. The meta-heuristic algorithms use the fitness function values to evaluate how good the binary test sequences are. It is important to mention that the binary search is performed as follows: During the first iteration, the initial binary test sequences are generated in a pseudo-random way. Then, the Device Under Verification is verified using these input sequences. After that, the coverage information is processed and the Compact-BinDE or PSO algorithms calculate the fitness value to evaluate the sequence, and a new possible solution is generated. One advantage of the algorithm is that only saves the best current solution for the next iteration, thus reducing the memory requirements. The algorithm 7 presents the steps of the test vector generation method used in this research (using Compact-BinDE algorithm).

Algorithm 7: General method used to perform the generation of test vector sequences

```
Configuration of the Device Under Verification (DUV).
Initialization of variables in the testbenches
Configuration of the verification modules.
Meta-Heuristic algorithm initialization.
while Stop condition is not met do
    while Convergence criteria is not met do
        Generate new test sequences based on the PV vector values.
        If the criteria is met, then the BinDE mutation and crossover are applied. If the
        criteria is not met, then a binary Local search is performed.
        Evaluate the device.
        Analyze the functional coverage information.
        Evaluate the fitness function based on the set of coverage points.
        Save the current best solution.
    end
end
```

To configure the size of the test vector sequences, it is necessary to review the functional device specification, the size of the connections between the device and the vector generator

module, as well as the testbenches information. In this research the configuration of the length and command fields for every test sequence was reconfigured in a manual way.

The Verification process of our proposal involves several modules, the blocks diagram of this process is shown in Figure 5.1. A set of vector sequences is generated at the beginning of the process. The coverage points are sampled during the device simulation. Each evaluation of a set of sequences is performed by the ModelSim simulator over linux OS. Once the functional verification is finished, the coverage information is analyzed. This analysis determines how good the sequence is. Afterwards, a new sequence is generated. The obtained results are reviewed based on the functional specification. This method uses the coverage information obtained at the end of the simulation. This information is carried to the directed automatic test generation module which decides if the sequence will be saved or not, and generates a new sequence for the device. The process is stopped when a determined coverage percentage is reached or when a specific number of iterations is met.

The representation of the test sequences (possible solutions) is based on a configurable fixed-length binary string, which represents a set of test vectors for the Device Under Verification. Therefore, the best solution is a set of vectors which produces the best functional coverage percentage. For example, in the case of a FIFO memory, each sequence can be composed of a sequence of 256 vectors of 8 bits to test the data registers. This sequence is based on the number of used data in the instructions verified. Moreover, the command fields were configured in manual way based on the functional specification. The equation 5.5 shows the representation of the sequences.

$$P_i = \{\vec{v}_1, \vec{v}_2, \vec{v}_3, ..., \vec{v}_n\} \tag{5.5}$$

where:

$$\vec{v}_i \subseteq X, X \equiv \{0,1\}^8 \tag{5.6}$$

In the case of the UART bus device, the command fields of the sequences were configured in manual way in order to disable invalid sequences during the functional verification. For example, a reset command is composed by 8 bytes where every field includes an specific value to restart the device. An example of a binary sequence in order to write binary data at the device is a sequence which is composed by 270 bits where 11 bits are used by write command fields and 259 by binary data field. In order to check the data written it is necessary to use a read sequence

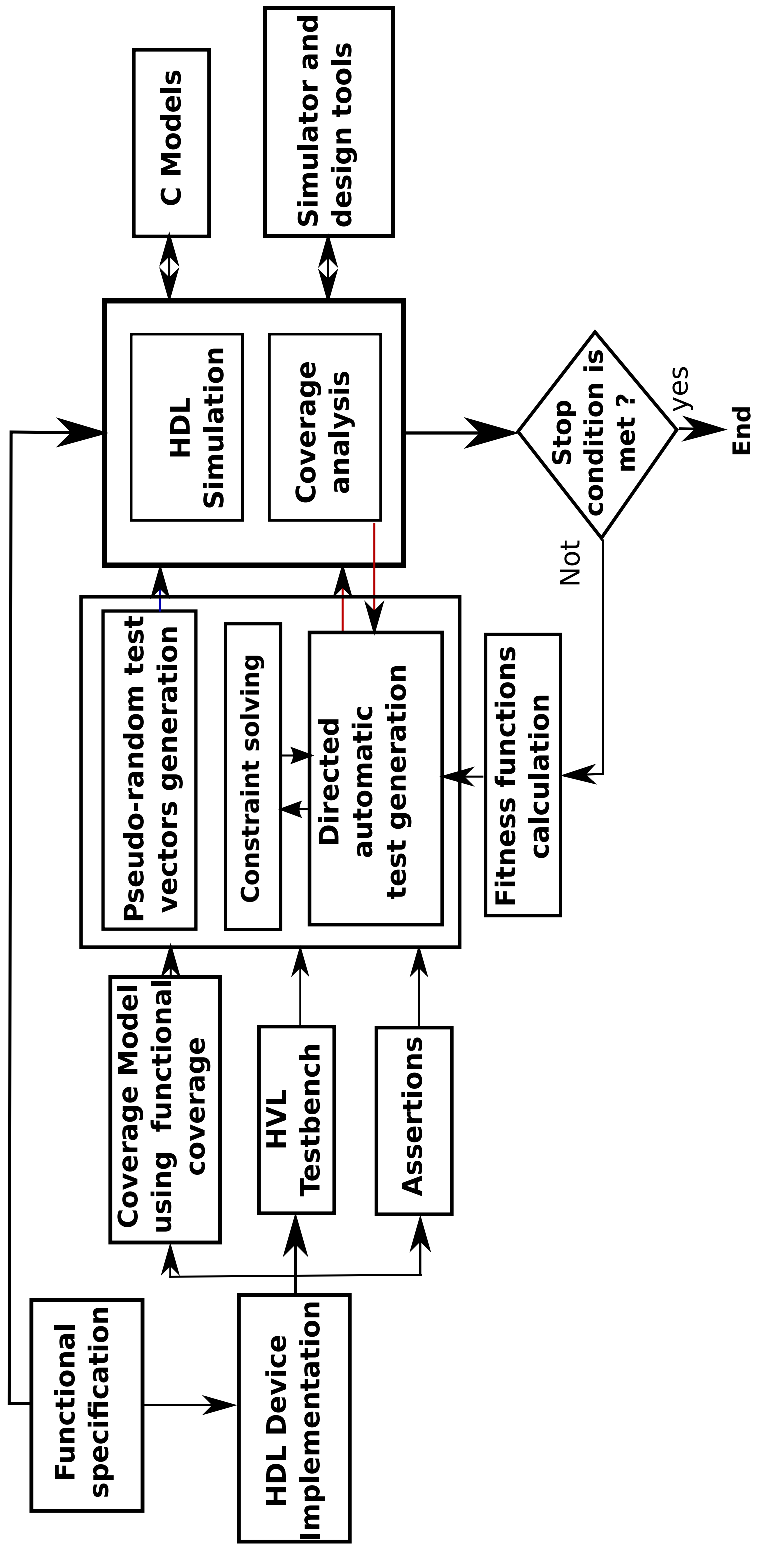


Figure 5.1. Blocks diagram of the test sequences generation method using the CoverPoints information as a coverage metric.

where 11 bits are used by read command field and 259 by data. In the next subsection the fitness functions are defined. The fitness values are required to guide the search during the generation process.

5.2.1 Fitness functions

Different holes are produced when the functional verification is applied. Giving the same weight to all test cases can be a bad strategy because if the fitness function is based only on the total percentage obtained, then, the weight of the easy and hard cases will be equal. Due to this, sequences which cover common points and sequences which cover hard cases will not show significant differences. This means that the convergence of the algorithms will be slow, producing a long simulation time. Even, in the case of a device with small complexity of functionality the simulation time can represent tens of minutes. Due to this, it is necessary to focus on the points with more holes to build the fitness functions using the information obtained from the all points. The first step is to cover most of the points using the total percentage. The second step is to cover the sets of CoverPoints with most of the holes, maximizing their coverage.

The first fitness function is described by the Equation 5.7 which is defined as the reciprocal of the percentage of holes to the set of coverage points. In this case, the fitness values are focusing to cover the maximum number of current holes in that set of points. This is an example of a fitness function using the not covered points (holes) percentage for a specific points set.

$$f_1 = MAX(\frac{1}{P_{eh}}) \tag{5.7}$$

Where:
P_{eh} is the coverage percentage of holes in a specific set of points.
The second fitness function is defined in the equation 5.8. It includes the percentage of current covered bins for all points and the not covered points in specific sets. The fitness function is defined as the reciprocal of the sum of the percentage of bins not covered P_{eh} in a specific set of CoverPoints and the percentage of covered bins for the total number of coverage points P_{Tc} in the *DUV*.

$$f_2 = MAX(\frac{1}{P_{eh} + P_{Tc}}) \tag{5.8}$$

Where:
P_{eh} = percentage of not covered bins in specific points.
P_{Tc} = percentage of current covered bins in all points.
Where:

$$P_{eh} = \sum_{i=1}^{n} \frac{bins\ \ non\ \ covered\ \ in\ \ specific\ \ points}{total\ \ number\ \ of\ \ bins} \tag{5.9}$$

$$P_{Tc} = \sum_{i=1}^{n} \frac{number\ \ of\ \ bins\ \ covered}{total\ \ number\ \ of\ \ bins} \tag{5.10}$$

In the next subsection the proposed verification platform is described. The designs and the algorithms were verified using the connection designed for functional verification between C language and SystemVerilog with the implemented designs.

5.2.2 Verification platform

The verification platform was constructed by an interface between C and SystemVerilog which connects the DUV with the verification environment and C files used for the different algorithms interacting for the test generation method. Different steps are performed during the evaluations in the platform. When the simulation finishes, a set of statistic information is generated. This information includes the obtained coverage percentages from the simulation, the number of evaluations, the best, the worst, and the average coverage percentages, the configuration of the parameters, the number of errors, and the best solutions in each experiment. All this data is saved in text files.

Figure 5.2 shows a block diagram of each module of the verification system used to perform the experiments. The first block shows the Design Under Verification (DUV). It is implemented using Verilog language. To perform the functional verification process a test environment in SystemVerilog was implemented and connected with the DUV. Then, a functional coverage model was proposed using the functional specification and the device implementation. This model was implemented in SystemVerilog language. The next block represents the simulator tool. In this case the Modelsim v6.5 simulator was used to perform the device simulations.

When a simulation is made, the coverage information is retrieved, analyzed and sent to the

Compact-BinDE algorithm (particle swarm optimization or binary genetic algorithms). Then the fitness function value is obtained based on the coverage information and assigned to test sequence. Finally, the best sequence is saved.

Evaluation of coverage points is performed according to the times in the specification of the device. These values are taken from the test generation module after simulation of the device. A monitor module is used in order to review that data obtained at the output is according the expected behavior.

The simulation platform has the necessary times in order to wait for the evaluation of every device test. Also, it is necessary that functions are contained by the platform to test the device each time by the test sequences algorithm generation.

Statistic files are saved into text files by the software platform. Figure 5.3 shows a schema of the designed software platform in order to perform the functional verification process. The connection between the device and the C language module was made using a ”Direct Programming Interface (DPI) interface”. The verification environment uses the interface to communicate with the external modules, control the signals and deliver the information to the device.

Verification platform is composed by the following modules:

- Test vector generation module. This module contains the evolutive algorithms (genetic algorithm, compact differential evolution algorithm and particle swarm optimization algorithm). The functions of this module are: calculate the fitness values, analyze the possible values and produce a new test sequence.
- Monitor Module. This module is composed by monitors which check the inputs and outputs of the device under verification. When the functional verification is made, the data obtained is reviewed in order to check the correct functionality.
- Statistic module. It saves the statistic information which is obtained from the simulation of the device. The data includes: coverage information, best, worst and average solution, configuration parameters among others.
- Connection interface. Connects the coverage model and implementation with verification environment and the algorithms.
- Coverage models and implementation module. In the first case, the coverage model represents the functionality by mean coverage points. This model is used in order to review if the implementation meets the functional specification.

In the next subsection the way to apply the evolutive algorithms over the proposed platform is

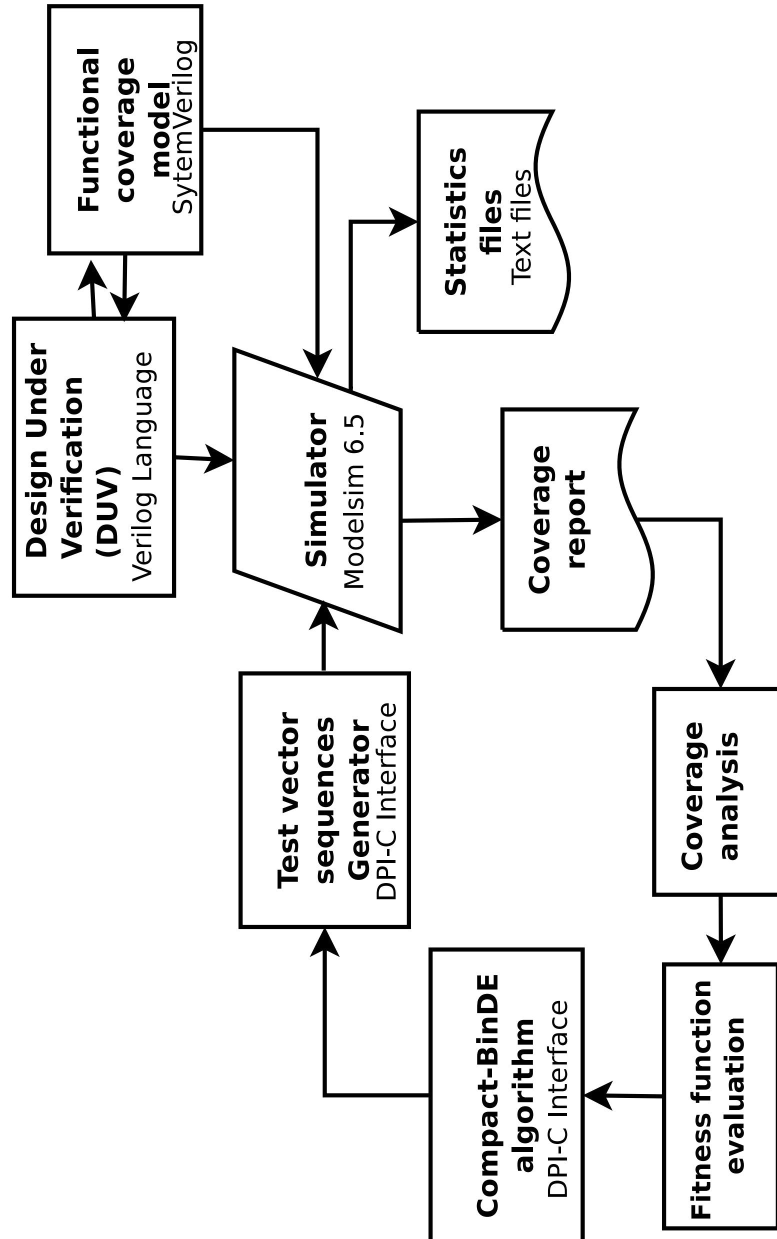


Figure 5.2. Verification interface using the proposed Compact-BinDE algorithm.

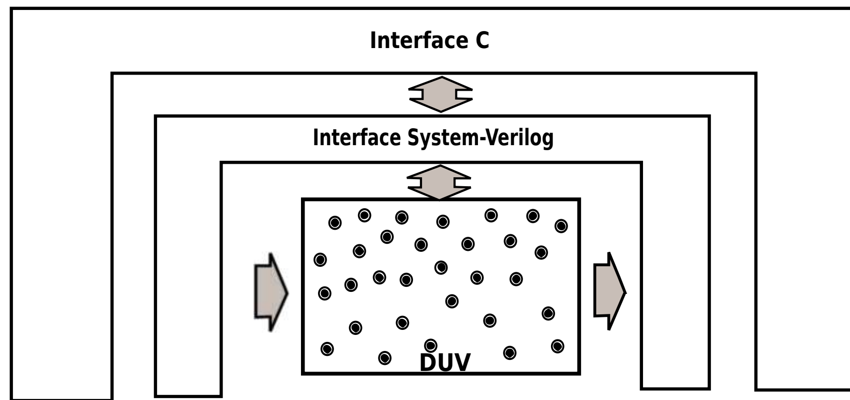


Figure 5.3. Schema of the designed platform in order to perform functional verification.

described. In this research the experiments were made using two different devices (A type FIFO memory and a UART bus ip core). An objective of this research is to propose a new method to generate test sequences maximizing the functional coverage using the PSO and Compact-BinDE algorithms.

5.2.3 Applying meta-heuristic algorithms over the proposed platform

The meta-heuristic algorithms were implemented over the proposed platform using the C language. Every binary sequence was introduced at the the device input according to the correct times controlled by the C-SystemVerilog interface. Different from other applications using evolutive algorithms, the total time required to evaluate every test sequence depends directly of the time used to make the device simulation and the computing resources. For instance, an evaluation of a medium size device can require hundreds of milliseconds. Therefore, the meta-heuristic algorithms need to optimize the resources making a good search over a big binary search space.

In this work we implemented the binary Genetic algorithm, binary Particle Swarm Optimization algorithm and a binary version of Differential Evolution algorithm. These meta-heuristic algorithms were included in the Directed Automatic Test generation module of the proposed platform. In order to use each meta-heuristic algorithm a set of configuration parameters should be set.

Figure 5.4 shows the steps of the genetic algorithm used on the verification platform. In this case, every individual represents a set of test vector sequences. At the beginning, the individuals

(test sequences) which constitute the population are initialized, then, each of them are evaluated and chosen based on the fitness values. After that, the crossover and mutation operators are applied in order to produce the offspring. Finally, the best solution is compared with the other individuals of the population and the best is saved. The set of steps is performed again while a stop condition is not met.

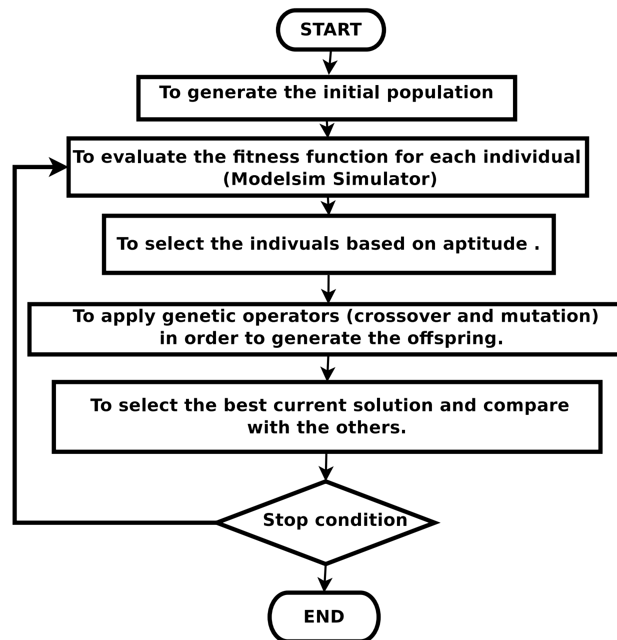


Figure 5.4. Flow Diagram binary Genetic algorithm algorithm used for generating sequences of vectors on the proposed verification platform.

On the other hand, the binary Particle Swarm Optimization algorithm was applied using the same proposed platform. Figure 5.5 contains the main steps of the PSO algorithm applied on the simulation platform for the device evaluation. For this algorithm each particle represents a test sequence for the device which maximizes the functional coverage. At the beginning, the population constituted by a set of particles is initialized. After that, every particle is evaluated and the fitness values are saved. These fitness values are compare with their last fitness values and the best position for each particle is saved. The best solution and the new velocity for each dimension (bit position) of every particle are calculated. Finally, if a stop condition is not met a new iteration is repeated.

Each time the design is evaluated, and the algorithms review the number of holes that

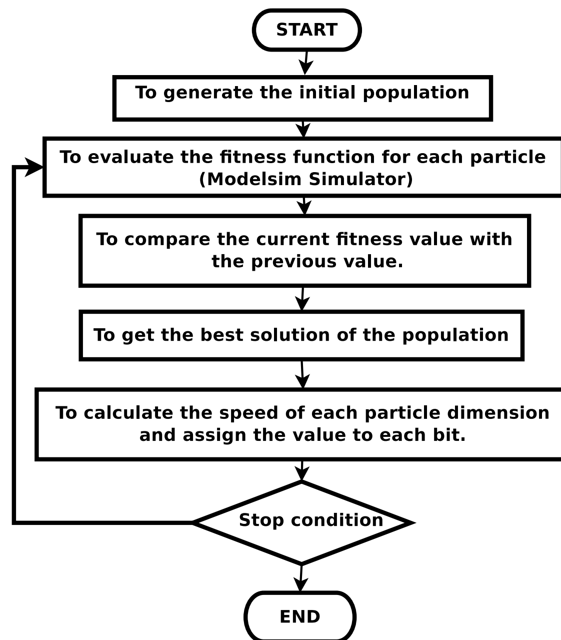


Figure 5.5. Flow Diagram PSO algorithm used for generating sequences of vectors.

exist and the points that have already been trained by the fitness function. These find the best sequences for such values and likewise generates better functional coverage values.

Moreover, the proposed compact binary differential evolution algorithm was implemented in the directed automatic test generation module. Figure 5.6 contains the main steps of the Compact Differential Evolution (Compact-BinDE) algorithm which is used in the simulation platform in order to evaluate the devices. Each vector represents a set of test sequences at the device input which maximizes the functional coverage. When simulation is made, the coverage percentage is feed-backed. This information is analyzed and delivered to the test generation module. After that, the Compact Differential Evolution algorithm obtains the fitness function values based on the coverage information; then it generates new test vector sequences. In the next subsection the modeling of the digital systems will be shown. To do this we use the coverage points and the functional specification of the devices.

5.2.4 Modeling the Device Under Verification

The design of digital systems can represent an art because the human intention is implicitly in the device implementation. It means that different implementations can meet the same expected

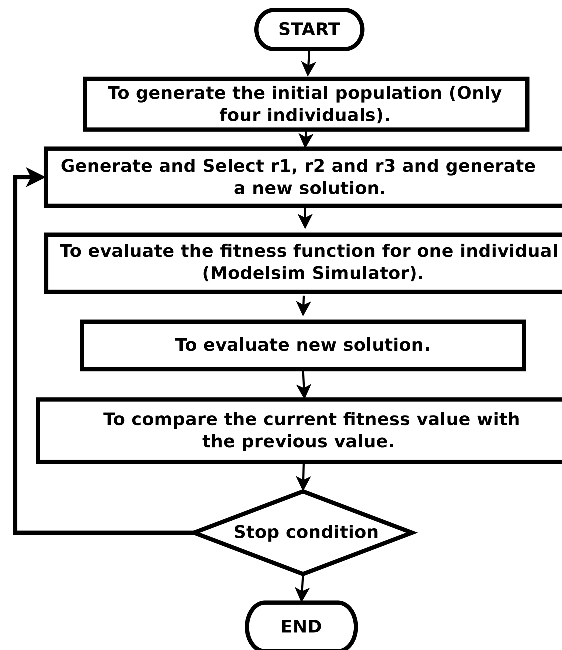


Figure 5.6. Flow Diagram Compact-BinDE algorithm used for generating sequences of vectors.

behavior which was proposed in the original specification. These implementations are according to the human criteria and the hardware resources.

In this research, we use functional coverage models based on the main events (CoverPoints) of the device implementation. It means that the models are based on the original proposed intention (functional specification). Figure 5.7 shows a UART bus and a coverage model obtained from the specification. In this case, every signal represents a CoverPoint which is included in the Coverage Model. Therefore, the set of CoverPoints represents the coverage model. Each of these points contains the relevant events that allow monitoring the behavior of the digital circuit during the injection of test sequences under different scenarios. The coverage model is connected with the implementation and the monitor module which checks if every condition is correctly exercised through the clock cycles. Both are simulated in the software platform. The results generated by the device are compared with the expected results based on the functional specification.

The implementation of coverage models was made in SystemVerilog language, which allows combining the hardware implementation with other languages such as Verilog, SystemC, etc. In the case of the coverage model of the UART bus device it contains 785 bins, which are monitored

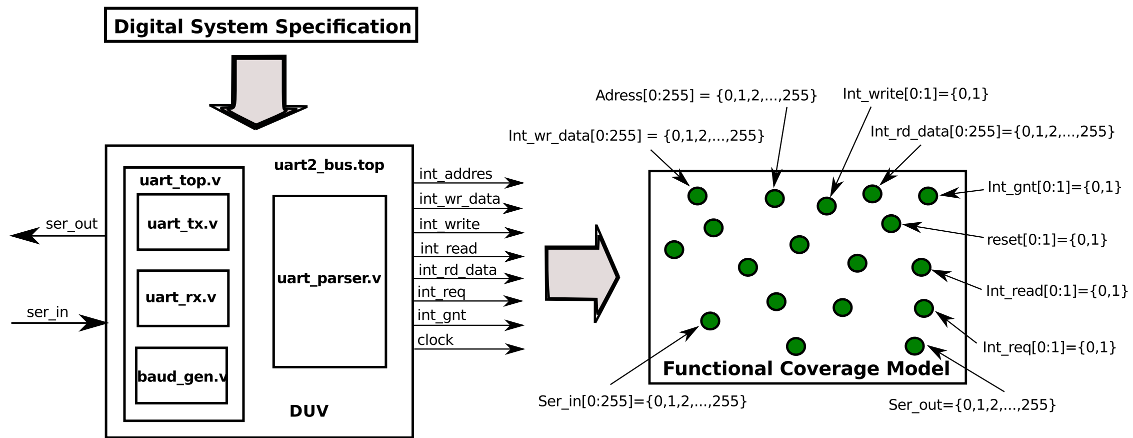


Figure 5.7. Model representation of a Device Under Verification (UART-bus) using coverage points to describe the behavior.

and stored in coverage reports. Moreover, the values of each sequence are stored at the end of each device evaluation.

5.2.5 Calculating and analyzing Functional Coverage

It is important to mention how the Directed Coverage was implemented is this research. In general, the test cases and the analysis of the coverage information are developed by hand because after the device evaluation the coverage reports are saved and analyzed in order to focus on the current holes and write new test vectors.

Figure 5.8 shows the Directed Coverage Functional Verification which is performed in a manual way. In this case, the verification engineer maps the functional specification and the implementation in order to build a functional coverage model. After that, a set of test cases are written by the engineers. Then, the device under verification is simulated by means of a software tool and the coverage information is saved into files. Then, the information is analyzed in order to check which CoverPoints have been covered and which points need to be covered representing holes in the verification.

Different from the last verification method, in this research the proposed platform performs the Directed Coverage Functional Verification process in an automatic mode. In this case, the model is proposed in a manual way; however, the test vectors are generate by means of the evolutive algorithms. Moreover, the analysis of the coverage information is reviewed and used

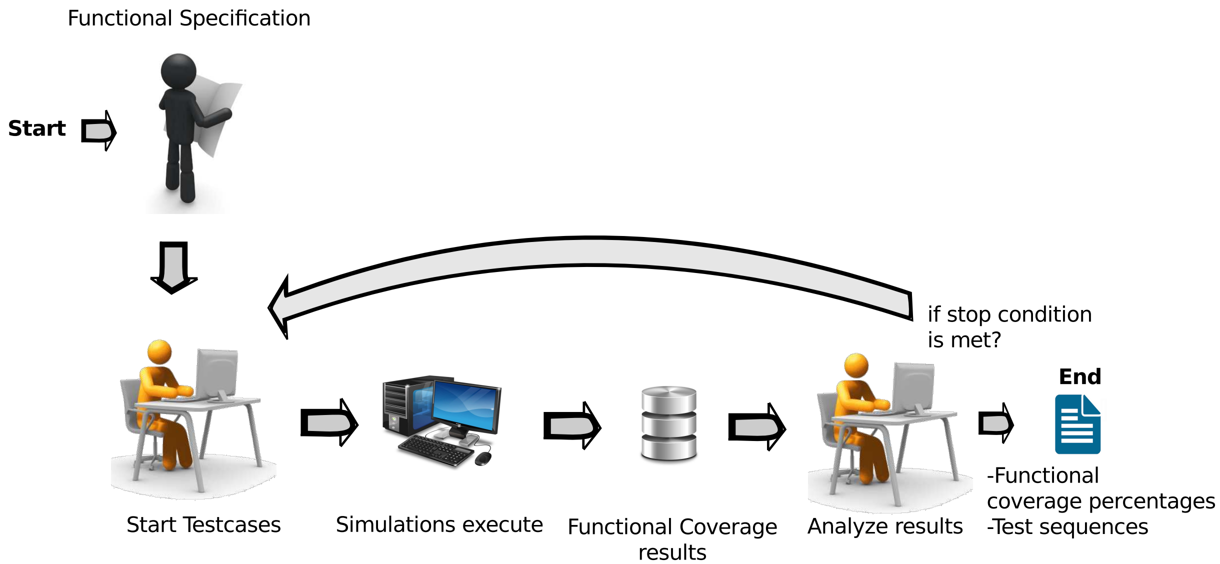


Figure 5.8. Manual process to perform the Directed Coverage Functional Verification.

in order to calculate the fitness function. After that, the meta-heuristic algorithms take the fitness value of every possible solution and generates a new test sequence to deliver it at the device input. Figure 5.9 shows the Automated Directed Functional Verification process used in this research.

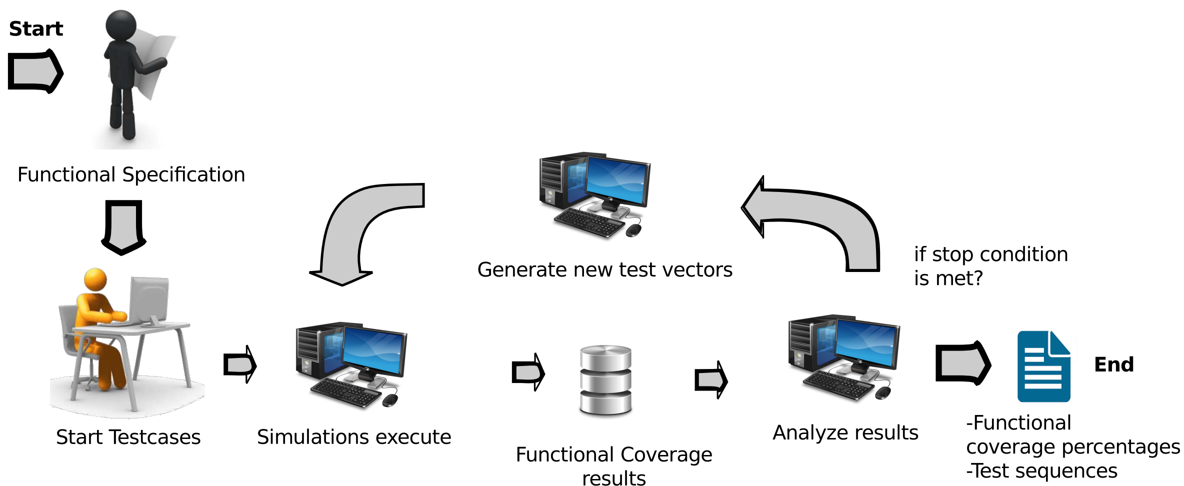


Figure 5.9. Automated process to perform the Directed Coverage Functional Verification.

Important features of the proposed system are the automatic device evaluation and the calculating and evaluating of coverage information according to the proposed fitness functions at the beginning of the process. These values are used to evaluate the binary test sequences and determine the search for new sequences to increase the coverage values.

In order to discard the case where the all points have the same weight, the points were divided in different sets. The CoverPoints were divided in sets with different weights; For example, the sets of points $S = p_1, p_2, p_3, ..., p_n$ can be adjusted with a vector of weights $H = w_1, w2, w3, ..., w_n$ in order to focus in the main Coverpoints. Therefore, when the device is evaluated the functional coverage percentages will be represented by most of the points included in the specific set. Equation 5.11 was used in order to calculate the functional coverage for the CoverPoints. The coverage is used to calculate the fitness functions after simulations.

$$Cg = \frac{\sum_i W_i * p_i}{\sum_i W_i} \tag{5.11}$$

In the next subsection the HDL simulation will be described, also, main characteristics about the configuration for the devices will be commented.

5.2.6 HDL Simulation of the Device Under Verification

The control of times and data transfers between the device and the verification environment is performed by a software tool (modelsim v6.5 simulator). A digital system can contain very complex modules working at the same time. The modules process different signals. Due to this, the evaluation of the operations needs to be monitored and saved in a correct way using appropriate software tools.

The first step was to combine the coverage model, device implementation and each of the verification environment modules. After that, the implemented test-benches inject the sequences at the digital system input from the binary vector generation module.

The Modelsim v6.5 software from Mentor Graphics company was used to simulate the digital systems. Another simulator tool could be used with the restriction of interface compatibility (Direct Program Interface) DPI. This interface allows programs to combine high level languages like C, C ++, Matlab, among others with hardware description languages (SystemC, SystemVerilog, Verilog) that allow interacting with the implementation of the digital system. The set of modules that comprise the meta-heuristics and other algorithms as well as text files were implemented in C language unlike the verification environment which is implemented

in SystemVerilog and Verilog languages.

Figure 5.10 shows the implemented process of Directed Coverage Functional Verification. The schema is composed of coverage models based on the functional specification. These models are connected with the device implementation by mean of the verification environment. Therefore, when the test sequences are injected at the device the coverage information is delivered to the coverage analysis module which selects the points values and finally, the fitness values are calculated. The values are used to direct the search to the interesting points, and at the same time, covering the characteristics in order to maximize the total functional coverage. The interaction between the device simulation and the meta-heuristic algorithms represents the hybrid method to generate the test sequences.

Synchronization of the signals time intervals for driving the DUV were controlled using the values according to the specification of each device. Such control allows injecting the binary sequences while verification environment monitors the coverage points. The main task here is to simulate the device behavior under the required scenarios at each clock edge.

Figure 5.11 shows the graphical interface of the Modelsim simulator used for the evaluation of the device test vectors. Each sequence of vectors is introduced into the device in each simulation. Different from manual simulations the device is automatically verified. The simulator tool exercises the DUV by means of different signals through the clock signals and the changes of the input data.

5.3 Resume

This chapter describes in detail the proposal for the test vector generation method. The method is based on the application of meta-heuristics to generate test vector sequences that maximize coverage values obtained by performing the functional verification process. This process is known as directed coverage functional verification.

In this research a new binary differential evolution algorithm based on binary domain operators as well as on the principle of compact genetic algorithm was described. The algorithm contains few configuration parameters, which allows applying it in different scenarios making small changes in the configuration and needs few memory resources required for implementation in a hardware device. In addition, other meta-heuristics are used, such as: the binary genetic algorithm and particle warm optimization algorithm. These algorithms were implemented in the test generation module.

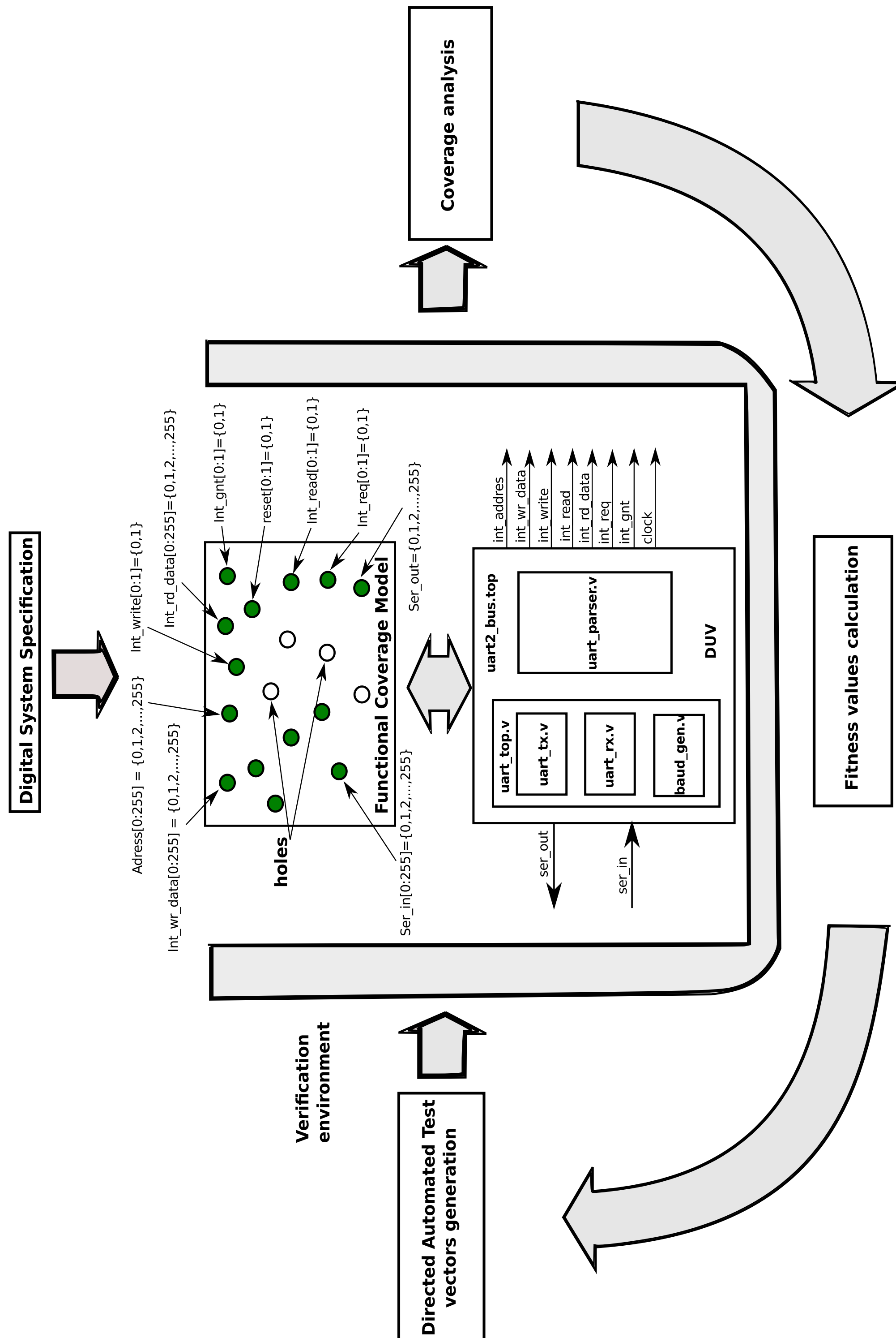


Figure 5.10. Implemented process to perform the Directed Coverage Functional Verification.

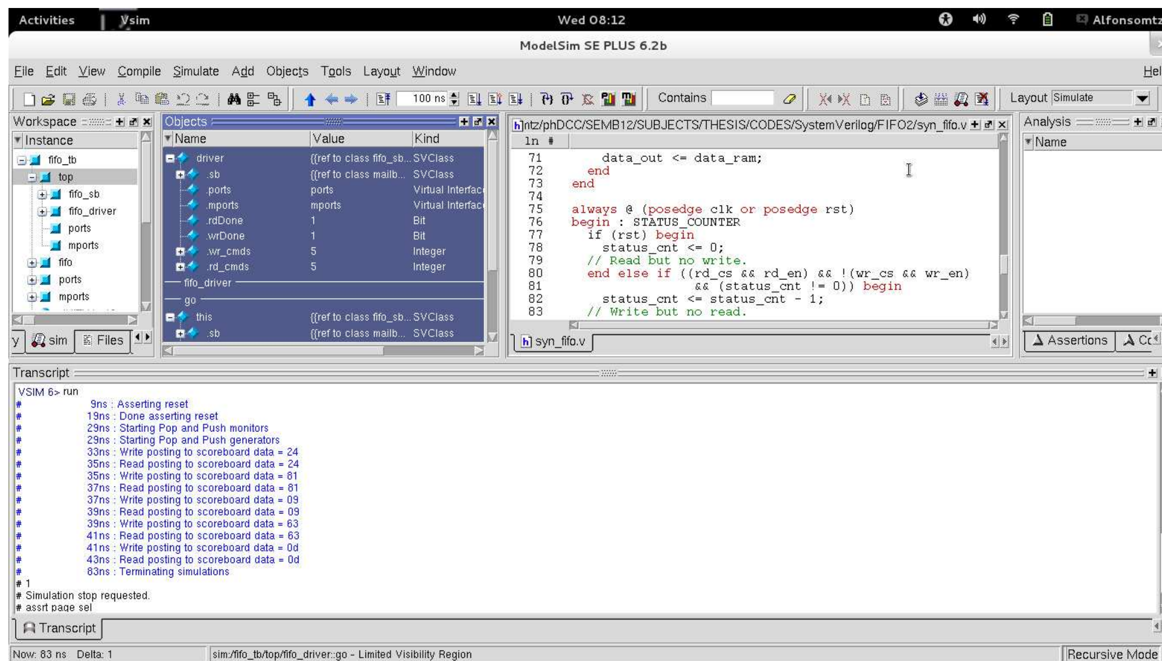


Figure 5.11. Modelsim simulator used in the experiments.

In addition, different aspects of the proposed method were described among which are: The steps in the sequence generation during the device evaluation, fitness functions used, the modeling of hardware devices as well as the simulation and analysis of results at the end of the simulation.

A software platform was implemented for functional verification. This software platform uses an interface for connecting the device under verification, verification environment and external modules containing algorithms and analysis functions coverage obtained. These modules are responsible for conducting the process of generation of tests aimed at coverage.

In the following chapter, experiments and results obtained using the proposed scenarios with different configurations of parameters are presented. The main contribution is the application of compact-BinDE and PSO algorithms and proposed fitness functions in order to maximize the coverage values.

CHAPTER VI

Experiments and Results

Chapter 6

Experiments and Results

In order to show the performance of the proposed test generation method we take two different devices (FIFO memory and UART-IP bus core). Different from problems where the evaluation consumes short times the functional verification can require more than tens of milliseconds to evaluate the hardware devices. Even devices of small size can require bigger times than problems of software. The delay generated by thousands of evaluations represents a considerable difference with respect to other problems.

The device behavior can be expressed by set of variables, events, expressions. It means, these variables can take a set of values in order to perform functions described in the functional specification. The test sequences at the input of the device generate changes of the signals and then exercise the device under verification. In this context, the relationship between the input sequences and the events (CoverPoints) is not trivial. Also, due to the device implementation can affect the holes produced after the verification. To focus in set of points can improve the performance of the verification and reduce the number of iterations of the algorithms.

In this chapter, the case study as well as the experiments will be described in detail. In the first section the devices for functional verification and the characteristics of the proposed coverage models are given. After that, the configuration settings and the characteristics of the computer equipment which was used to perform the experiments are presented. In the next section, the results obtained using the compact differential evolution algorithm (Compact-BinDE). After, experiments using the Genetic algorithm, Particle Swarm Optimization are described. Then the results of the comparison with a genetic algorithm, a pseudo-random test generation, Compact-BinDE and a Particle Swarm Optimization algorithm are shown. Finally, there is a discussion of the results.

6.1 Case Study

This case study was developed to show the main characteristics of the Compact-BinDE algorithm and Particle Swarm Optimization algorithms. It is important to mention that different from the conventional genetic algorithms, the Compact-BinDE includes the compact genetic algorithm principle based on the statistical values of each of the elements of the population. Also, the elitism strategy is used, which outperforms the results obtained when using the basic genetic algorithms [70]. The added differential evolution principle allows us to produce new solutions which approach a global solution by means of the difference of two sequences added to a third sequence chosen from the population. One advantage of the compact-BinDE algorithm is that it allows us to save only three or five individuals, independent of the dimentionality of the problem, thus reducing the memory requirements even if the problem has a high dimentionality.

In this research two different devices were used to analyze the performance of the algorithm. To perform the functional verification a UART bus IP core and a synchronous FIFO memory were used. These devices were obtained from the Opencores page. They were implemented in Verilog language and the testbenches were designed in SystemVerilog. According to the platform described in 5.2.2 the configuration and the experiments were made using the functional specification. The experiments were performed using the proposed verification platform on the Fedora Core Linux operating system.

The first device is a Synchronous FIFO which has a bidirectional bus and chip select, write enable, read enable, output enable, empty flag, and full flag signals. In this case, for the FIFO memory, 784 bins were used from the coverage points which include: address, input data, output data, rd_en, wr_en, full flag, empty flag, wr_cs, rd_cs. Although the complexity of this device is low, it is a common device which can be part of more complex devices such as UARTs, processors, etc.

The second device consists of a UART bus IP device, which is a converter of a UART to bus IP core. This core can be used during initial board debugging or as a permanent solution when high speed interfaces are not required. The internal bus is designed with an address bus of 16 bits and a data bus of 8 bits. The core contains a UART transmit block and a receive block which share a common baud rate generator and a command parser. The parser supports two modes of operation: text mode commands and binary mode commands. Text mode commands are designed to be used with a hyper terminal software and enable easy access to the internal bus. Binary mode commands are more efficient and also support buffered read and write operations

with or without an automatic address increment. Using the functional specification and the implementation of the UART bus IP device, 785 bins were used from 12 coverage points. The stimulus time and clock signals were configured according to the specification and successful initialization of this device. In the next section the experimental setup is described.

6.1.1 Experimental Setup

To analyze the performance of the algorithms, we proposed a set of experiments using different scenarios. In this research, "scenario" means a set of configured parameters for the Meta-heuristic algorithms. According to each experimental section, every scenario was run a number of times. Independent from the scenario, the configuration and initialization of every device were performed using the functional specification for each one.

For each scenario, different parameters were changed, i.e. for the Compact-BinDE algorithm: crossover CR probability value, probability value P_{ls} for local search, the population size, number of LS evaluations for local search, total number of evaluations and, the crossover and mutation DE functions. In this research "mutation" means a change in each new generated solution using a pseudo-random value, accomplished by adding the difference of two sequences to another sequence, chosen according to the structure of the Compact-algorithm as was explained in 5.1. Also, two fitness functions which were introduced in the 5.2.1 were used to evaluate the test sequences during the experiments. The proposed algorithm uses these functions to evaluate how good a possible solution is based on the coverage data obtained in the set of chosen points.

To present the obtained results, different values were analyzed, including the best coverage percentage, the worst coverage percentage, the average coverage, and total time of the simulation. Also, in each section of the experiments, different constraints were changed. For example, we only used one test sequence as a constraint in one section and we used two sequences in another, which provides an increase in the length of possible test sequences.

Experiments using the same devices in the proposed verification platform were performed in order to make the comparison between the Compac-BinDE algorithm and other algorithms, such as pseudo random test generation and the general version of the genetic algorithm. The objective was to show the performance of the algorithms searching test sequences which cover all coverage points. The same fitness functions were used to do the experiments. Modifying the different parameters of each algorithm, different results are obtained, however, the PSO and

Compact-BinDE algorithms reached the best coverage values in most of cases. In the case of the genetic algorithm, the following parameters were modified: population size, crossover and mutation percentages, and number of evaluations. Due to the fact that the genetic algorithm is a meta-heuristic based on population, it requires the evaluation of all individuals or all possible solutions in each iteration, which produces a slow convergence.

To perform the experiments the following characteristics of the computer used were: Fedora Core 18 Linux operating system, Processor : AMD Athlon(tm) II X3 440, cache size: 512 KB, cpu: 3 GHz. RAM memory: 4Gb. The simulator used was: Modelsim version 6.5. In the next subsection the set of experiments is described. Different scenarios were used and the best scenarios are presented.

6.1.2 Experiments and Results with Compact-BinDE Algorithm

In this section, we proposed a set of experiments using different scenarios and devices. In this research a "scenario" means a set of configured parameters for the algorithms. According to each experimental section, every experiment was run a number of times. Different values are shown including the best coverage percentage, the worst coverage percentage, the average coverage, and total time of the simulation. Also there are different constrains used in each section, for example, we used only one test sequence as a constraint in on section and we used two sequences in another, which provides an increase in the length of possible test sequences, etc.

In the first section, a FIFO memory was used as a Device Under Verification (DUV). The experiments were realized taking one test sequence of 2048 bits as the length of each possible solution. This constraint reduces the possible options considerably. Because of the reduction of possible options, a higher number of evaluations is required. The experiments were made changing the following parameters of the Compact-BinDE algorithm: The number of total individuals (population size), the type and the CR probability value of BinDE crossover, the total number of evaluations and the P_{ls} percentage value for the applied local search.

The Table 6.1 shows the parameters for 4 best scenarios using different Differential Evolution structures. Each column represents a set of parameters for different DE structures (/rand/1/bin, /best/1/bin, /best/2/bin, /rand/2/bin). The first row shows CR probability value of BinDE crossover. The population size is shown in the second row, and the P_{ls} values are shown in the third row. The stop criteria was 15000 evaluations during the verification process. Each

experiment was run 30 times for each scenario.

Table 6.1. Parameters of four different scenarios using Compact-BinDE algorithm

Compact-BinDE / Parameters	/rand/1/bin	/best/1/bin	/best/2/bin	/rand/2/bin
CR value	0.00065	0.00065	0.00010	0.00010
Population size	500	500	500	500
P_{ls}	0.002	0.002	0.0025	0.0025
Iterations	15000	15000	15000	15000

Table 6.2 shows the obtained results for each scenario which were described in Table 6.1. The first column shows the number of scenarios used in order to perform the generation of test sequences. The second column shows the total number of evaluations used. The best and the worst coverage values are shown in the third and fourth columns respectively. The fifth column shows the average coverage percentage calculated from the 30 runs for each scenario. Finally, the total time used to perform the verification process is shown in the sixth column. The algorithm reached values over 99 percent coverage in most of the experiments with 15,000 evaluations as stop criteria. Reviewing the obtained results, we can see that the second scenario produced the best coverage percentage running the same configuration 30 times.

Table 6.2. Coverage values obtained with four different configurations of the Compact-BinDE algorithm

Scenario	Evaluations	Best value	Worst value	Average	Time (min)
1	15000	99.023	96.875	98.059	1,688
2	15000	99.414	97.265	98.164	1,675
3	15000	99.218	95.70	98.092	1,697
4	15000	98.828	97.460	98.196	1,717

The second part consisted of running experiments using different population size. In this case for the experiments, the $DE/rand/1/bin$ version of the Compact-BinDE algorithm was used. Table 6.3 shows the set of parameters for four of the best scenarios used. The stop criteria

was 15000 evaluations. Different population sizes were used for each scenario: 450, 500, 600, 800. Different CR crossover values were used and each scenario was run 30 times.

Table 6.3. Parameters of four best scenarios using /rand/1/bin version of the Compact-BinDE algorithm with different population size

Parameters \ Compact-BinDE	/rand/1/bin	/rand/1/bin	/rand/1/bin	/rand/1/bin
CR value	0.00065	0.00060	0.0006	0.0005
Population size	450	500	600	800
P_{ls}	0.0005	0.0005	0.0005	0.005
LS evaluations	1	1	1	1
Iterations	15000	15000	15000	15000

Table 6.4 shows the obtained results for each scenario of Table 6.3. The total number of evaluations, the best, the worst, and average coverage percentage values, and the average of the simulation time are presented in the second, third, fourth, fifth and sixth columns respectively. In this case, the first and the fourth scenarios produced the best average values of percentage of coverage. In the first scenario, this reached 99.218% using 450 individuals.

Table 6.4. Values of Coverage percentage obtained with different population size

Scenarios	Evaluations	Best value	Worst value	Average	Time (min)
1	15000	99.218	96.875	98.092	1.67
2	15000	99.609	96.484	98.170	1.68
3	15000	98.828	96.484	98.053	1.66
4	15000	99.218	96.093	98.19	1.67

If the length of each possible solution is increased to two input test sequences (4096 bits), then the performance increases considerably, and the algorithm reduces the total time needed to obtain test sequences to cover all coverage points. The set of parameters for the four best scenarios is shown in Table 6.5. In this case the stop criteria was 5000 evaluations.

Table 6.6 shows the obtained results with the scenarios presented in 6.5. In the first column the average of the number of evaluations for each experiment during the 15 runs is shown,

Table 6.5. Parameters of four best scenarios using a solution length of two sequences

Parameters \ Compact-BinDE	/rand/1/bin	/best/1/bin	/best/2/bin	/rand/2/bin
CR value	0.0015	0.0015	0.0015	0.0015
Population size	500	500	500	500
P_{ls}	0.005	0.005	0.005	0.005
LS evaluations	5	5	5	5
Iterations	5000	5000	5000	5000

and the total best coverage value is shown in the second column. Finally, the average time (in minutes) is included in the third column. In these experiments the algorithm reached 100% with the functional coverage as coverage metric. Due to the increase of the length of sequences, the algorithm produced better percentages, thus reducing the time needed for the verification.

Table 6.6. Coverage values and total time for each scenario using different mutation functions for the Compact-BinDE algorithm

Average evaluations	Best value	Average Time (min)
1553.00	100.00	0.5248
1429.13	100.00	0.3645
1491.13	100.00	0.3820
1706.20	100.00	0.4373

Also, different scenarios were used to analyze the performance of the algorithm in more complex devices. In these experiments a UART-IP core device was used as a Device Under Verification using the Compact-BinDE algorithm. Table 6.7 shows the parameters for the four best scenarios using different configurations (rand/1/bin, best/1/bin, best/2/bin, rand/2/bin) of the algorithm. The CR crossover values are shown in the first row, the population size values are presented in the second row. The P_{ls} percentages for the local search are shown in the third, and the number of evaluations for local search LS is presented in the fourth row. Finally, the number of evaluations is shown in the fifth row. In this case, the length of each test sequence

was reduced to one sequence of 2072 bits as a constraint. Each scenario was run 15 times. The stop criteria was 6000 evaluations.

Table 6.7. Parameters of four best scenarios using the UART-IP device

Parameters \ Compact-BinDE	/rand/1/bin	/best/1/bin	/best/2/bin	/rand/2/bin
CR value	0.00080	0.00050	0.00050	0.00060
Population size	600	600	600	600
P_{ls}	0.005	0.005	0.005	0.005
LS evaluations	5	5	5	5
Iterations	6000	6000	6000	6000

Table 6.8 shows the obtained results with the scenarios presented in Table 6.7 using the UART-IP core. The first column shows the number of scenarios and the second column presents the total number of evaluations for each scenario. The best, the worst, and the average coverage percentage are shown in the third, fourth and fifth columns respectively. Finally, the average time of simulation for each scenario is presented in the sixth column. In this case, the fourth scenario reached an average coverage value of 97.5%, using 122.63 minutes as a average time from 15 runs.

Table 6.8. Obtained results for four scenarios using Compact-BinDE algorithm

Scenario	Number of evaluations	Best value	Worst value	Average	Time (min)
1	6000	97.916	97.005	97.421	133.860
2	6000	98.177	96.223	97.291	136.002
3	6000	98.567	95.833	97.005	135.868
4	6000	97.786	97.135	97.578	122.634

Different scenarios were used to analyze the performance of the Compact-BinDE algorithm with an increase in the length of each possible solution. The length of each possible solution was increased to two sequences (4144 bits), the $f1$ fitness function for a set of points, and the stop criteria was decreased to 1500 evaluations with different CR crossover percentage values. The

scenarios were run 15 times. Table 6.9 shows different parameters for 4 best scenarios. Every column represents a different structure of the compact-BinDE algorithm. The crossover CR values are shown in the first row, while the second row shows the population size. The third row presents the probability value for the local search P_{ls}. The number of evaluations of the local search and the total number of iterations are shown in the fourth, and fifth rows respectively.

Table 6.9. Parameters of the Compact-BinDE using a length of two sequences with the UART-IP device

Parameters \ Compact-BinDE	/rand/1/bin	/best/1/bin	/best/2/bin	/rand/2/bin
CR value	0.0015	0.0015	0.0015	0.0015
Population size	600	600	600	600
P_{ls}	0.005	0.005	0.005	0.005
LS evaluations	5	5	5	5
Iterations	1500	1500	1500	1500

The obtained results are shown in Table 6.10. The number of scenarios is shown in the first column. The number of evaluations used as stop criteria is shown in the second column,while the best, the worst and the average coverage percentage values are shown in the third, fourth, and fifth column respectively. The average time is shown is the sixth column. According to the results, when the fitness values were focused in a set of points in most of cases the algorithm reached 100% using different scenarios, with average coverage values over 99%. When the length of the possible solutions was increased, the time was reduced considerably. In this case the fourth row shows the best average of coverage percentage using 29.216 minutes. The first and the fourth scenarios produced the best averages of coverage values.

The Compact-BinDE algorithm produced good solutions even though only one test sequence is used to cover the set of coverage points. When the length was increased to two or more test sequences, the total time was reduced considerably. The results show that averages of coverage percentages over 99% were reached with less time than when the length of one sequence was used for every possible solution. One improvement will be made by modifying the original coverage model and reviewing the verification environment after the verification. Figure 6.1 shows the coverage values obtained according to the number of evaluations using the f_1 fitness function and the DE/rand/1/bin configuration for the Compact-BinDE algorithm verifying the

Table 6.10. Results obtained using different configurations of the Compact-BinDE algorithm to test the UART-IP device

Scenario	Number of evaluations	Best value	Worst value	Average	Time (min)
1	1500	100.00	99.479	99.895	21.634
2	1500	100.00	99.348	99.661	16.588
3	1500	100.00	96.223	99.895	17.877
4	1500	100.00	99.869	99.921	29.216

UART-IP core. Six hundred individuals, and 6000 evaluations were selected as stop criteria. In this case, it is important to say that the constraint used was a length equal to one sequence for every possible solution. According to the obtained results, after 3000 evaluations the coverage percentage was over 95%, and the percentage was around 98% when the stop criteria was reached.

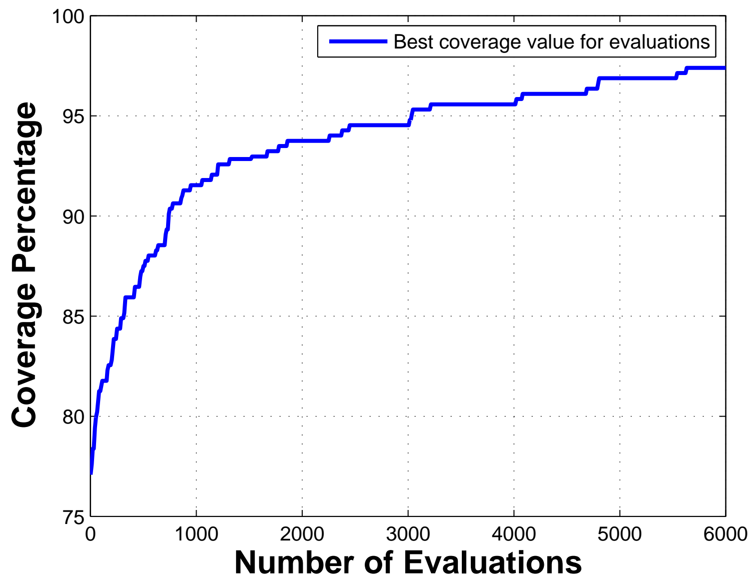


Figure 6.1. Coverage values obtained according to the number of evaluations using the f_1 fitness function and the DE/rand/1/bin configuration for the Compact-BinDE algorithm and the UART-IP core.

Figure 6.2 shows a comparison of different configurations of the Compact-BinDE using the

UART-IP core as Device Under Verification. According to the graph, the coverage percentage values are increased in different ways when the crossover value CR is changed. In the first 2000 evaluations there are different coverage values, but, after that, the coverage percentages reach values over 98% of coverage.

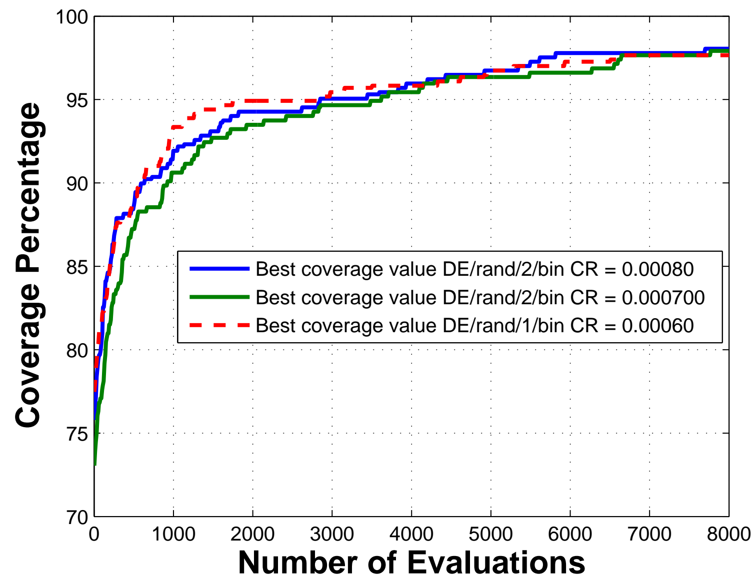


Figure 6.2. Comparison of the Coverage Directed Test Generation using different configurations.

Figure 6.3 shows the coverage percentage using a length equal to 2 sequences (4,144 bits) for every possible solution with the Compact-BinDE algorithm. The horizontal axis shows the number of evaluations used to obtain the coverage percentage values which are shown in the vertical axis. In this case, the configuration for the algorithm was DE/rand/1/bin.

The comparison between different crossover CR values using the DE/rand/1/bin configuration is shown in Figure 6.4. According to the obtained results, the scenario with $CR = 0.0016$ converged quickly. Even in this case, the scenario using $CR = 0.0014$ converged more slowly but this reached 100% before the others. The CR value determines the degree of the mutation and crossover applied to produce a new solution. This factor can help to increase the speed of the convergence of the algorithm, but it can also get stuck on a local solution. The same parameters of the local search were used for the three scenarios, in which 100% was reached in the first and second cases.

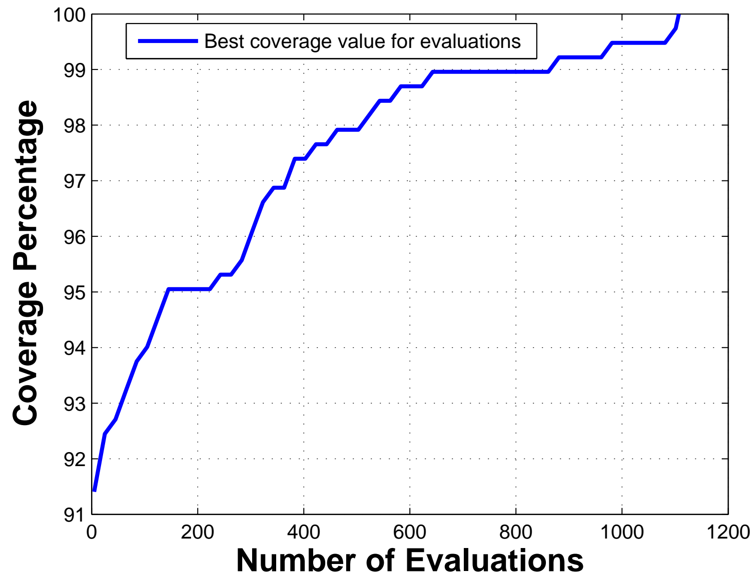


Figure 6.3. Coverage Directed Test Generation percentages for different iterations number using 2 sequences and a CompactBinDE/rand/1/bin configuration.

Also, the comparison between $f1$ and $f2$ fitness functions was performed using the UART-IP bus core. Figure 6.5 shows the comparison between $f1$ and $f2$ fitness functions using the DE/rand/1/bin configuration for the Compact-BinDE algorithm. The horizontal axis shows the number of evaluations and the vertical axis represents the coverage percentage values. According to the obtained results, using fitness function $f1$, the algorithm obtains better coverage percentages than fitness function $f2$ and the convergence is reached more quickly in most of cases. The reason is that the values obtained from fitness function $f1$ are inversely proportional to the set of generated holes during the functional verification. It means function $f1$ focuses on the set of generated holes on run time. Other fitness functions can be proposed to improve the performance of the algorithm. In the next section a Comparison between the Compact-BinDE algorithm, a general version of the binary genetic algorithm, and a pseudo-random generation will be presented.

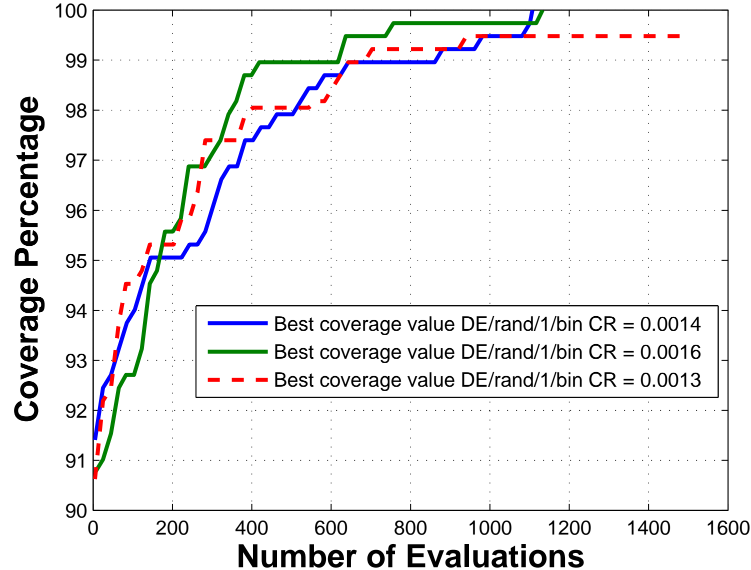


Figure 6.4. Comparison of Coverage Directed Test Generation with different CR values using 2 sequences and a CompactBinDE/rand/1/bin configuration.

6.2 Experiments and Results with other Meta-heuristic Algorithms

In addition, to compare the performance of the compact differential evolution (CompactBinDE) algorithm, other algorithms were used. The genetic algorithm, the Particle Swarm Optimization algorithm and the pseudo-random generation were used in the proposed platform. These algorithms were implemented on the proposed platform and different parameters were set to configure the different scenarios.

Beside, the proposed test generation method uses the Particle Swarm Optimization algorithm in order to verify the devices. The binary version of PSO algorithm was proposed by Keneddy [60]. This version was implemented on the proposed verification platform. Moreover, the simple version with elitism of Genetic algorithm was implemented to make a comparison with the other algorithms.

For the experiments with meta-heuristics $f1$ fitness function was used. Sequences of 4144 bits were used to verify the UART device. Also, test sequences of length 4096 bits were used

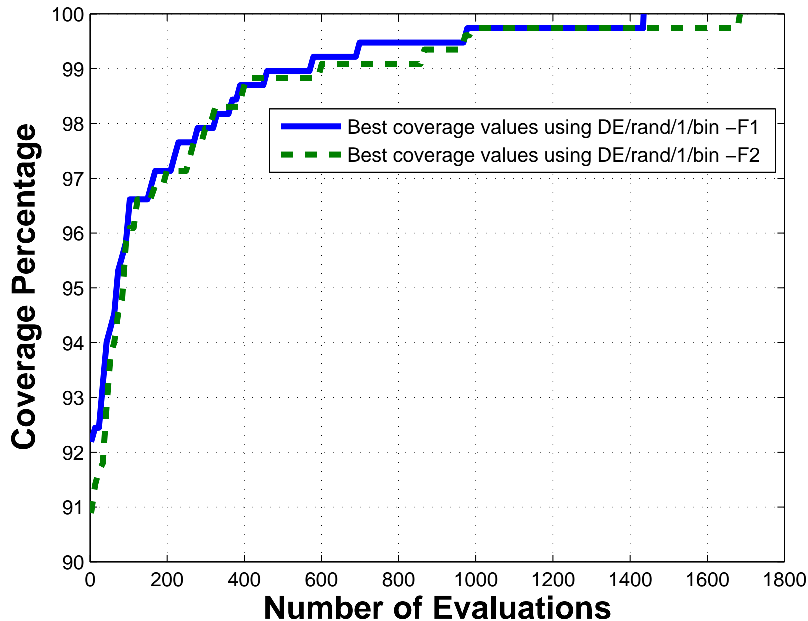


Figure 6.5. Comparison of Coverage Directed Test Generation using 2 different fitness functions $f1$ and $f2$ and sequences of 4144 bits as the length of each possible solution, and CompactBinDE/rand/1/bin configuration.

to verify the FIFO memory. Different scenarios were set to analyze the results and get the best scenarios based on the above conditions. In the case of the genetic algorithm population sizes, crossover and mutation percentages were changed. In the case of the PSO algorithm, the globalbest and localbest topologies were implemented with different scenarios and the results were compared. The same coverage models and devices were used for all algorithms. The next subsection presents the experiments and results obtained using the genetic algorithm.

6.2.1 Experiments using a binary Genetic algorithm

Also, the genetic algorithm was implemented in the proposed platform. Different experiments were performed using different scenarios. In each scenario, the population size, crossover and mutation percentages were changed. The UART device and FIFO memory were used as devices under verification. Also, 30 runs were made using $f1$ fitness function and the number of evaluations as stop criteria. According to this, the experiments were made and some

of the best scenarios are presented in this section.

Table 6.11 shows the parameters of the genetic algorithm to verify the FIFO memory. Each column represents a set of parameters for 3 best scenarios. The first row contains the crossover percentages which were set to 0.5 in the three cases. The mutation percentages are shown in second row. The third row shows the population sizes. A stochastic universal sampling was used as type of selection in all cases, this is shown in the fourth row. Finally, the number of evaluations is shown in the fifth row. For these experiments 30 runs were made with 5000 evaluations as stop criteria.

Table 6.11. Parameters for experiment using binary Genetic algorithm for FIFO memory

Parameters \ Values	1	2	3
Crossover Percentage	0.50	0.5	0.5
Mutation Percentage	0.0012	0.0018	0.0015
Population size	100	200	125
Type of selection	stochastic universal sampling	yes	yes
Number of Evaluations	5000	5000	5000

Table 6.12 shows the results obtained using the given scenarios from Table 6.11. Each column shows the coverage values obtained using each scenario. The best, the worst, and the average of coverage percentages are given in the first, second and third rows respectively. Average of the number of evaluations and average time in minutes are given in the fourth and fifth rows.

Table 6.12. Results obtained using the binary Genetic algorithm for FIFO memory

Final values	Scenario 1	Scenario 2	Scenario 3
Best value	99.60	96.67	98.43
Worst value	94.92	92.77	94.33
Average value	97.29	94.86	96.41
Average evaluations	5000	5000	5000
Average Time (min)	1.079	1.071	1.065

According the obtained results, when 100 individuals were used as population size, the algorithm reached higher coverage percentages than bigger populations. Using the same number of evaluations the average value of coverage percentages for using this scenario was 97.29%. Even the algorithm not reached the convergence with 5000 evaluations, in this scenario the algorithm reached coverage percentages higher than 97%. Also it was observed that mutation percentages produced different results even without change of other parameters. For example, using values of: 0.0012, 0.0015, 0.0010 the coverage percentages were close using all other parameters equal. On the other hand, when 125 were used (third scenario) the average of percentages around 96.41% were reached. Different from the last cases, in the second scenario using 200 individuals the algorithm covered the Coverpoints slower than the other scenarios. Is important to do mention that the number of iterations depends on the number of test sequences from the chromosome (short lengths). It means, if less test sequences are chosen to test the device, then the algorithms can need more iterations to search good solutions.

On the other hand, the genetic algorithm was used with different configurations in order to verify the UART device, thereby, experiments with each of the different scenarios were performed. Table 6.13 shows the values of the parameters for three best scenarios using the genetic algorithm with $f1$ fitness function. Each column shows the parameters for each scenario. The crossover percentages are shown in the first row, the second row shows the mutation percentages, the population sizes are shown in the third row, finally, the type of selection and the number of evaluations are given in fourth and fifth rows respectively. In these experiments, test vector sequences of 4,144 bits were used. 30 runs for every configuration were performed and the stop criteria was 6000 evaluations.

Table 6.13. Parameters for experiment using binary Genetic algorithm

Parameters \ **Values**	**1**	**2**	**3**
Crossover Percentage	0.5	0.5	0.5
Mutation Percentage	0.001	0.0015	0.0003
Population size	150	200	100
Type of selection	stochastic universal sampling	yes	yes
Number of Evaluations	6000	6000	6000

Table 6.14 shows the results obtained in each of the scenarios. The first row shows the best

coverage value obtained in all experiments for each scenario, the second row shows the worst coverage value and the third row shows the average of the best coverage percentages obtained in all runs; in the fourth row the average of the number of evaluations is shown, the fifth and sixth rows show the average time value in minutes and hours respectively. For this set of experiments the third scenario presented the best results. The best coverage value obtained was 99.73% with an average of coverage of 98.78% and an average time around 246.94 minutes (4.1 hours). The results showed that one hundred individuals were needed to reach percentages over 98%. One disadvantage is the increase in simulation time due to a greater number of individuals. In the case of the first and second scenarios the coverage percentages were less than third scenario. Using the first scenario, coverage percentages around 97.5% were reached. When the second scenario was used lower coverage percentages were obtained. Also, the total time required to make each experiment was over 4 hours. The total time required is directly proportional to the number of iterations, the simulation tool and computing resources used. In the next subsection experiments using the Particle Swarm Optimization algorithm will be described.

Table 6.14. Results obtained using the binary Genetic algorithm

Final values	Scenario 1	Scenario 2	Scenario 3
Best value	99.479	98.43	99.739
Worst value	95.833	95.05	96.875
Average value	97.513	96.727	98.78
Average evaluations	6000	6000	6000
Average Time (min)	245.127	254.807	246.94
Average Time (hrs)	4.085	4.246	4.115

6.2.2 Experiments using a Binary Particle Swarm Optimization (PSO) algorithm

The Particle Swarm Optimization algorithm was used to perform the experiments using the UART bus and the FIFO memory. Different scenarios to test the method were used in the

experiments, velocity, number of particles and number of neighborhoods were changed. The topology of PSO algorithm was *lbest* and *gbest* using the $f1$ fitness function for each scenario, 30 runs were performed.

In the fist part a FIFO memory was tested using the proposed platform. Table 6.15 shows the configuration parameters of the PSO algorithm for three of the best scenarios using test sequences of 4096 bits in order to verify a FIFO memory. Each column shows the set of parameters for each scenario. The first row shows the number of particles, the number of neighborhoods is shown in the second row, the maximum and minimum velocity for each particle are shown in the third and fourth rows, the fifth row shows the maximum value of ϕ (positive random number), the type of topology, the number of experiments and the stop criteria are given in the sixth, seventh and eighth rows respectively.

Table 6.15. Algorithm parameter values used for PSO scenarios with FIFO memory

Parameters	Scenario 1	Scenario 2	Scenario 3
number of particles	9	9	3
number of neighborhoods	3	3	1
Velocity max	8.0	8.6	8.5
Velocity min	-8.0	-8.6	-8.5
ϕ max	2.0	2.0	2.0
Topology	lbest	gbest	gbest
Number of experiments	30	30	30
Stop criteria	5000	5000	5000

Table 6.16 shows the average of coverage percentages, the average of the number of evaluations and the average time obtained using the three scenarios from Table 6.15. In this case, test sequences of 4096 bits were used to test the FIFO memory. Reviewing the results, these showed that in most of cases, the best coverage percentage values obtained with these parameters were around 100%. Using 5000 generations as a top criteria. The Particle Swarm Optimization (PSO) algorithm had better performance when the velocity of particles were between 6.0 to 9.0 because the algorithm converged quicker when the particle velocity is kept in those ranges.

Experiments were also conducted using the global version of PSO algorithm and a UART bus core with a stop criteria of 6000 evaluations. Table 6.17 shows the parameter values for

Table 6.16. Results obtained for experiment 1 using a FIFO memory

Final values	Scenario 1	Scenario 2	Scenario 3
Average value	100	100	100
Average evaluations	4960	3811	2092
Average Time (min)	1.121	1.30	0.5339

three of best scenarios. For these sets of parameters 30 runs were carried out for each scenario. Each column represents a different scenario. The number of particles is shown in the first row. The second row shows the number of neighborhoods, the maximum and minimum velocities are shown in third and fourth rows respectively. The maximum ϕ value is shown in the fifth row. Finally, the number of evaluations is shown in the sixth row.

Table 6.17. Parameters for Particle Swarm Optimization algorithm using a UART device

Parameters \ Values	Scenario 1	Scenario 2	Scenario 3
Number of Particles	16	16	9
Number of neighborhoods	4	4	3
Max velocity	9.0	8.5	8.5
Min velocity	-9.0	-8.5	-8.5
ϕ max	2.0	2.0	2.0
Number of evaluations	6000	6000	6000

Table 6.18 shows the results obtained using the three scenarios of the PSO algorithm from Table 6.17. The best, the worst and the average of coverage percentages are shown in first, second, and third rows respectively. The average of the number of evaluations is shown in the fourth row. Finally, the total times in minutes and hours are shown in fifth and sixth rows respectively.

Using test sequences of 4,144 bits, the algorithm reached average of coverage values over 98%. It was seen as increasing the velocity parameter and the convergence was quicker. One of best scenarios has a velocity over 8.5 and 9 particles with 3 neighborhoods.

Table 6.18. Results obtained using the Particle Swarm Optimization algorithm with a UART bus ip core

Final values	Scenario 1	Scenario 2	Scenario 3
Best value	100.0	100.0	100.0
Worst value	98.95	98.95	100.0
Average value	99.80	99.71	100.0
Average evaluations	5757.86	5764	3697.83
Average Time (min)	236.38	233.25	155.81
Average Time (hours)	3.93	3.88	2.59

It is important to mention that the Particle Swarm Optimization algorithm generated levels over ninety nine percent of functional coverage when this used a set of 9 particles with 3 neighborhoods and a length of 4,144 bits. In the case of sixteen particles, percentages reaching more than ninety eight percent from six thousand algorithm generations as stop criteria were obtained. The results showed that at the beginning of the algorithm execution, percentages of over eighty percent were quickly reached. This is due to initializing the particles pseudo-randomly which generated different test vectors covering many bins of the coverage model. After that, the particles move very quickly over functional verification space. In the next subsection a comparison between the algorithms will be presented.

6.2.3 Comparison between algorithms

Experiments were performed using the pseudo-random generation, genetic algorithm, particle swarm optimization and Compact-BinDE algorithm. In the first part, 30 runs were made in order to test the FIFO memory. In the case of the pseudo-random generation, test sequences were generated based on the coverage information obtained in every iteration. After every iteration if a new generated sequence covers new points then it replaces the current best sequence. For the Genetic algorithm the parameters were set as follow: Population size = 100, Mutation percentage = 0.0012, Crossover percentage = 0.50, and a length of 4096 bits. In the case of PSO algorithm the parameters were: Number of particles = 3, Neighborhoods = 1, V_{max} = 8.5, V_{min} = -8.5, ϕ = 2.0, Topology = gbest. For the Compact-BinDE algorithm the parameters were set : CR= 0.995, P_{ls}= 0.0025, Population size = 400 and a local search =

pseudo-random. After run the experiments, the coverage percentages obtained were reviewed and the best scenarios were analyzed.

Table 6.19 shows the obtained results using the last parameters of the algorithms. Each column represents different parameters of the algorithm. The best, worst and average of coverage values are shown in the first, second and third rows respectively. The average number of evaluations and the average time in minutes are given in the fourth and fifth rows. According the results, the genetic algorithm reached better results than the pseudo-random generation. In the case of the Compact-BinDE algorithm, in most of the scenarios it reached higher coverage percentages than the pseudo-random generation and genetic algorithm. On the other hand, using the Particle Swarm Optimization algorithm percentages around 100% were obtained. In the best scenario, the algorithm reaches high coverage percentages reducing the total time used to perform the number of evaluations.

Table 6.19. Results obtained using the pseudo-random generation, binary Genetic algorithm, the Compact-BinDE algorithm and PSO algorithm to test FIFO memory

Final values	pseudo-random	Binary GA	Compact-BinDE	BinPSO
Best value	94.14	99.60	100.0	100.0
Worst value	91.79	94.92	98.04	100.0
Average value	92.74	97.29	99.63	100.0
Average evaluations	5000	5000	4717.66	2092
Average Time (min)	1.05	1.079	1.065	0.5339

In the second part, experiments using different scenarios were run 10 times for each generation method. The Device Under Verification used was an UART-IP core. Table 6.20 shows the parameters for the best binary Genetic algorithm scenario using 5000 evaluations as stop criteria. The fitness function was $f1$, and a length of 2 input test vectors (4,144 bits) was used. The first row shows the crossover percentage for each possible solution. The second row shows the mutation percentage, which, in this case means the percentage of change for each element of a sequence by means of a pseudo-random number which determines that probability value. The population size, type selection and maximum number of evaluations are presented in the third, fourth and fifth rows respectively.

In each case, 5000 evaluations were performed to reach the stop criteria. Table 6.21 shows

Table 6.20. Parameters for experiment using binary Genetic algorithm

Parameters \ Values	Binary GA
Crossover Percentage	0.5
Mutation Percentage	0.0003
Population size	100
Type of selection	stochastic universal sampling
Number of Evaluations	5000

the comparison of the results obtained using the Binary Genetic algorithm, the pseudo-random generation, and the Compact-BinDE algorithm algorithms. The best, the worst, and the average of coverage values obtained during the experiments are presented in first, second and third rows respectively. Finally, the average of the number of evaluations, the average time (minutes) and average time in hours are shown in the fourth, fifth and sixth rows, respectively. The results showed that in the best scenario the Compact-BinDE algorithm obtained 100% using fewer evaluations than the other two algorithms. The values of the parameters were changed in the experiments, which showed that the results obtained with the binary Genetic algorithm were very competitive, and in the best scenarios, using the Compact-BinDE algorithm better coverage percentages were obtained using less the number of evaluations.

Table 6.21. Results obtained using the binary Genetic algorithm, pseudo-random generation and the Compact-BinDE algorithm with the UART

Final values	Binary GA	pseudo-random	Compact-BinDE
Best value	98.95	95.83	100.0
Worst value	96.09	94.66	97.91
Average value	98.12	95.10	99.30
Average evaluations	5000	5000	4910
Average Time (min)	218.31	207.36	203.73
Average Time (hrs)	3.63	3.45	3.39

In the case of the pseudo-random generation, different coverage points were covered very

quickly because the generation of the test sequences is less complex, and so the total generation time is good. However, it was observed that when certain coverage percentages are reached, the algorithm improves the coverage percentages very slowly, which means that the algorithm repeatedly covers the same sets of coverage points, thus producing a waste of time in the simulation. Different from the pseudo-random generation, the proposed Compact-BinDE algorithm searches new test sequences based on the best current sequence, and it saves the values of each bit position of test sequences using the compact genetic algorithm and differential evolution principles. Due to this, the algorithm can produce better results with fewer evaluations than needed using pseudo-random generation.

The binary Genetic algorithm reached better results than the pseudo-random generation. The Genetic algorithm is based on the population because it takes advantage of a variety of different solutions to produce the new population. In the experiments, different scenarios were used for the binary genetic algorithm. According the results, the binary Genetic algorithm reached solutions over 96%. in most cases. One advantage of the Genetic algorithm with respect to other generation methods, is that it uses fewer computer resources to perform the generation of test sequences. However, when the Compact-BinDE algorithm was used, higher coverage percentages were reached in the best scenarios. The Compact-BinDE algorithm only replaces one test sequence of the population if, and only if the new test sequence is better. Also, the best directions direct the search to better sequences, and the algorithm covers more holes every time. Therefore, high coverage percentages can be reached very quickly.

In order to compare the four algorithms using UART-IP bus core different scenarios were used. 30 runs were made for every scenario of each algorithm. The pseudo-random test generation was implemented into the proposed verification platform. The genetic algorithm was used with 100 individuals, 0.5 as uniform crossover percentage, a mutation percentage of 0.0003 and a selection using stochastic universal sampling. The PSO algorithm was used with 9 particles, 3 neighborhoods, V_{max} = +8.5, V_{min} = -8.5, ϕ_{max} = 2.0, W_{max}=1.0, W_{min}=-1.0. The parameters of the Compact-BinDE algorithm were: CR = 0.995, P_{ls} = 0.0025, Population size = 450, Number of evaluations = 6000, version = Rand/1/bin, local search = pseudo-random. Table 6.22 shows the obtained results. Each column shows the results for each algorithm. The best, worst and average of coverage percentages are shown in the first, second and third rows respectively. Average of the number of evaluations is shown in the fourth row. The fifth row shows the average time in minutes and the average time in hours is given in the sixth row.

The Compact-BinDE algorithm can generate sequences for other devices using few

Table 6.22. Results obtained using the pseudo-random generation, binary Genetic algorithm, the Compact-BinDE algorithm and PSO algorithm with a UART device

Final values	pseudo-random	Binary GA	Compact-BinDE	BinPSO
Best value	95.833	99.73	100.0	100.0
Worst value	94.791	96.87	99.21	100.0
Average value	95.182	98.78	99.84	100.0
Average evaluations	6000	6000	5502.74	3697.83
Average Time (min)	255.15	246.94	223.41	155.81
Average Time (hrs)	4.252	4.11	3.72	2.59

parameters, which can be changed for other constraints. Also, due to the ability of the algorithm to represent the population as a probability distribution, the algorithm samples some values from the probability vector to obtain new test sequences. Therefore, the algorithm can be implemented with less hardware requirements than a basic version of the binary Genetic algorithm.

The results show that in the best scenarios the CompactBin-DE algorithm reached better results than the Genetic algorithm and the pseudo-random generation using a lower average of the number of evaluations. Therefore, the algorithm can be a good choice for generating test sequences to perform the functional verification of devices.

On the other hand, the Particle Swarm Optimization algorithm produced higher coverage percentages than pseudo-random and Genetic algorithms. Also in the best scenarios the PSO algorithm reached higher coverage values than the Compact-BinDE algorithm. Using different scenarios, the algorithm can produce coverage percentages over 98%. During the experiments different scenarios were used. Changing the population size and the number of neighborhoods the algorithm changed the performance. Depending of the length of sequences different number of iterations were needed in order to reach high coverage percentages.

6.3 Discussion

We can discuss the results obtained in the experiments on parts. First, according to the results, the Compact-BinDE algorithm reached high coverage values with appropriated fitness functions focusing on special CoverPoints on run time simulation. The strategy used increased

the number of covered points evaluating every possible solution with the fitness function. When the evaluation focused on the set of holes generated during run time, more functionality was exercised because the weight of the evaluation was higher for the points not full covered on run time.

Coverage percentages over 98% with different scenarios were reached using the Compact-BinDE algorithm. Also, the algorithm uses a local search to improve the binary search using the current best sequence. It is important to mention that when the length of test input sequences was increased, the average of coverage percentage obtained was over 99% for the best scenario, and the required time was reduced because the number of possibilities was increased. However, it could be interesting to search minimal lengths for the test input sequences which can test most of the coverage bins. Different from previous works, one objective of this research proposes one strategy to generate test sequences using the proposed Compact-BinDE algorithm.

Different population sizes were used in this research and the best scenarios were presented to show the performance of the test sequence generation method. The results show that the algorithm can generate good test sequences with an binary search selecting appropriate sequences during the functional verification during the run time. Moreover, a suitable number of individuals should be used to obtain better results with different scenarios. Independent of the dimentionality of the problem, the algorithm saves only a few number of individuals to obtain good results, thus saving computer memory resources. In the case of the number of individuals who are saved, the amount of individuals is different from the total number of individuals of the population. It means that different from the meta-heuristics based on population, the Compact-BinDE algorithm saves the statistical information of the total number of individuals into the PV vector. This information represents the best elements from all individuals who constitute the total population. To produce the offspring, the individuals are sampled from this PV vector and only some of them are saved in memory to produce the new individuals every time. For example, in the case of the basic configuration DE/Rand/1/bin, four individuals are saved in memory to produce a new individual every time using the crossover and mutation from the differential evolution principle. Therefore, the test generation method can be implemented in a device with reduced memory resources.

One advantage of the proposed algorithm is that few parameters can be modified according to the DUV and the constraints. The strategy is based on the current best sequence, and it saves memory for each bit position of the test sequences, which produces good results. On the other hand, setting different values of crossover probability (CR) and Local Search probability

P_{ls} with different scenarios, the algorithm can produce better coverage values than the pseudo-random generation and a basic genetic algorithm.

In the second part experiments using the genetic algorithm were performed. In this case, different parameters were changed: population size, crossover value, mutation percentage, number of evaluations. After that; three of the best scenarios were shown in the results section. In the case of UART device Average of coverage percentages over 96% was reached using population sizes between 100 and 200 individuals. Also, a FIFO memory was verified using the Genetic algorithm. In this case, percentages over 97% were obtained in one of best scenarios. Also, the evaluation time for each experiment was shorter than the evaluation time used by the UART device, around 1 minute using the computer resources described in section 6.1.1. This depends of the number of evaluations as stop criteria.

On the other hand, in the third part when the Particle Swarm Optimization algorithm was used to verify a FIFO memory using test sequences of 4096 bits and 5000 evaluations as stop criteria. Percentages around 100% were obtained in the best scenarios using the verification platform. Different scenarios were tested and it was observed that the global topology showed better results than the local topology. The algorithm showed good results using the global-best topology why the algorithm converged quicker covering 100%.

Also, the PSO algorithm was used with a UART-IP device. In this case the algorithm reached coverage percentages over 98% for best scenarios. In these scenarios the average of coverage percentages was around 99% using sequences of 4,144 bits and less than 6000 evaluations. The results were very competitive with respect to the Compact Differential evolution algorithm and genetic algorithm. Even in the best scenarios the Particle Swarm Optimization algorithm converged quicker. In order to control the convergence a re-initialization mechanism was used. Based on the current coverage percentage and velocity values of the particles. This produces the re-initialization of the velocity when the total coverage percentage is kept in a determined number of iterations. Using the FIFO memory percentages around 100% were reached for the best scenarios. Moreover, different scenarios were used, in a scenario with 9 particles and 3 neighborhoods the algorithm produced coverages of 100%. When different velocities were used the convergence was modified. In the next section the conclusions of this research will be presented.

Conclusions and future work

Chapter 7

Conclusions and future work

With the development of this research, good results were obtained using the Compact-BinDE and Particle Swarm Optimization algorithms. Using the devices (UART bus and FIFO memory) it was observed that the proposed method is an alternative for generating sequences of test vectors. Also, the use of fitness functions and coverage models can improve the performance and reduce the number of iterations required to achieve high functional coverage percentages.

The automated functional verification is performed by the interaction between the hardware implementations and the verification environment over a software tool. Due to this, a verification interface was designed in order to connect the devices under verification with other modules performing the directed coverage functional verification.

A Compact binary differential evolution algorithm was proposed. This algorithm is based on the Compact Genetic algorithm and differential evolution algorithm principles. Different from other versions, the algorithm uses binary operators for crossover and mutation. It means that the genotype representation is mapped directly to genotype in order to perform a binary search. So, statistical information of the population is used in order to reduce the memory requirements to implement the algorithm.

In this research, a binary Particle Swarm Optimization algorithm was implemented to generate test sequences for the devices. Different scenarios were used and the results were analyzed. The use of this algorithm produced competitive results with respect to pseudo-random generation, genetic algorithm and Compact-BinDE algorithm. Due to this, this algorithm represents a good alternative for test generation with respect to other algorithms.

In this chapter the conclusions obtained after conducting this research will be given. Further work for the improvement and application of the proposed test generation method for functional

coverages in digital systems is mentioned. Finally, the contribution of this research and publications in journals and international conferences will be presented.

7.0.1 Conclusions

In this thesis a test sequences generation method was developed in order to maximize the functional coverage on the functional verification of digital systems. The development and conclusions of this research can be summarized in different stages.

1.- The first was the design of a software platform, which allowed carrying out the interaction between the algorithms in a verification environment and device implementations. Different from the manual methods where human intervention is needed after the simulation is finished, in this case, the interaction is made in an automated way. This means, after the device simulation, the information is delivered, analyzed and the fitness values are calculated. Therefore, a new test sequence is injected into the device and the process is repeated until a stop condition is met. Moreover, the platform includes a set of options to configure the meta-heuristics, coverage models, device implementations, as well as various options for storing the statistics for the simulations and tests. This platform was used to perform the experiments and test the proposed method.

2.- The second stage involves the design of a module in order to configure the test-benches, coverage models, meta-heuristic algorithms and verification environment for the devices. For this, state-of-the-art current dynamic functional verification techniques were reviewed. Such techniques are based on device simulation and analysis of coverage information to produce new device evidence. This scheme was used to perform the simulation, monitoring, and reviewing of the information during the device stimulation. During the experiments, different problems were presented, such as clock synchronization and module configuration of the hardware devices. The characteristics of the devices were configured manually. The module was designed using C language and SystemVerilog. It allows configuration of the Genetic algorithm, Particle Swarm Optimization algorithm and Compact-BinDE algorithm. Also, other functions can be added in order to improve the proposed method.

3.- In the third stage, a new version of a compact differential evolution algorithm (Compact-BinDE) for binary representation was proposed. This is based on the compact genetic algorithm and the differential evolution principles. This algorithm uses the statistical

information for each position of binary strings and mutation and crossover operators between such sequences. Different from the binary genetic algorithm the Compact-BinDE algorithm uses the statistical information as population to produce individuals reducing memory requirements independently of the dimensionality of the problem. This algorithm was implemented in the proposed verification platform and tested with the two devices (FIFO memory and UART) using similar verification conditions to the other algorithms. The Compact-BinDE algorithm reached coverage percentages over 99% using a FIFO memory. Moreover, in the best scenarios it reached coverage values around 100%. In the case of the UART bus device the algorithm reached percentages over 98%, and for the best scenarios average percentages around 99% were reached. The results obtained were competitive with respect to pseudo-random generation, the genetic algorithm and a Particle Swarm Optimization algorithm.

4.- The fourth stage consisted of the implementation, test and comparison of the algorithms: genetic algorithm, Particle Swarm Optimization (PSO) algorithm and Compact differential evolution algorithm (Compact-BinDE). To do this, state-of-the-art binary versions were reviewed, since unlike the versions for real representation, binary versions allow us to directly map the phenotype to the genotype in the binary representation, which is implicit in the functional verification problem. These algorithms were tested using different scenarios in two different devices (UART bus and a FIFO memory). In the case of the UART when the genetic algorithm was used, functional coverage percentages over 96% were obtained. In the best scenarios average coverages over 98% were reached. According to the results, when the population size was increased, the convergence of the algorithm was slower in producing coverage percentages under the scenarios around 96%. Due to this, the number of evaluations was larger in order to increase the coverage than the Particle Swarm Optimization algorithm and Compact-BinDE algorithm.

In the case of PSO algorithm, there are some important aspects to mention. First, the global version obtained better results than the local version in most of cases. In the case of the global version, the influence of the best global particle in the population produced than the others moved quicker to the solution generating coverage percentages around 100% in a FIFO memory. In the best scenarios this algorithm shows better performance than pseudo-random generation and genetic algorithm, requiring less iterations to reach high coverage percentages. In the case of the UART device, coverage percentages over 98%

were reached. In the best scenarios coverage values of 100% were obtained. It was shown that when velocity values for every particle were increased, the PSO reached percentages over 94% very quickly. However, after that, more evaluations were required to reach higher percentages. Therefore, the results using Particle Swarm Optimization algorithm were very competitive with respect to the genetic algorithm and the Compact differential evolution algorithm.

5.- Finally, in this research two fitness functions were proposed in order to maximize the coverage percentages using meta-heuristic algorithms. According the results the $f1$ fitness function produced better results than $f2$ when it was used in the GA, PSO and Compact-BinDE algorithms. The fitness functions focus on sets of CoverPoints which are selected using the coverage obtained after device simulations. Also, the device behavior was modeled using coverage models. These models are composed by CoverPoints which were split in sets. It was observed from the experiments that focusing in all points the algorithms required more iterations to reach high coverage percentages. Due to this; different weights were assigned to the sets of points because when all points have the same weights the same cases can be covered a lot of times representing a waste in the simulation time.

There are different highlights of this research, between them, we proposed:

A new test sequences generation method based on Directed Coverage functional verification of digital designs. For this, a hybrid verification method philosophy was used, that is, the combination of verification based on simulation of the hardware devices and the use of meta-heuristic algorithms (Compact-BinDE, PSO algorithms) was made. Based on the results, it was observed that the meta-heuristics represent a feasible alternative in order to generate test sequences, these can be implemented to interact with the verification environment, and therefore to perform the functional verification of digital devices.

A new version of Compact binary differential evolution algorithm was proposed. This algorithm is based on the principle of compact algorithms, which unlike the algorithms based on the population, uses vectors with statistical values to represent the population, and it is not necessary to store in memory all individuals. The algorithm produced competitive results when applied to generate sequences of binary vectors. According to the results when the algorithm was used with different configuration scenarios, it was observed that the algorithm reached higher functional coverage percentages than 98% using an UART bus and a FIFO memory.

Compact meta-heuristics were used for first time. Such is the case of the compact differential evolution algorithm. The performance was competitive in comparison with other

meta-heuristic algorithms such as genetic algorithm and particle swarm optimization. Different from previous works, the main advantage of the compact meta-heuristic algorithms is that they require less memory resources for implementation. The compact differential evolution algorithm requires few configuration parameters and reduces memory consumption.

A verification platform was implemented to perform the functional verification process. This platform consists of modules for the use of the algorithms and the connection with the ModelSim Simulator v6.5. The coverage information and statistics results are stored for analysis and used in the verification process. This platform allows to implement coverage models as well as the algorithms for generating test sequences. Other simulation tools can be used with certain restrictions of support for the Direct Programming Interface (DPI) in order to connect the hardware description languages and C language.

A module for generating test sequences using meta-heuristic algorithms was implemented. This module allows setting the parameters for each algorithm and used in conjunction with the device under verification to be tested. The module interacts with the verification environment, analyzes the coverage values obtained, and evaluates each generated test sequence.

Coverage models were implemented for devices under verification and a strategy was used based on the division into sets of points focusing on the more relevant points and increase the values of total coverage quicker. In addition, fitness functions were proposed to evaluate how good is each solution.

7.0.2 Future Work

As future work of this research the following lines of research are proposed:

- The use of an extensive set of bench-marks in order to compare the meta-heuristic algorithms in functional verification of digital circuits.
- The combination of formal with meta-heuristic methods for the functional verification.
- Implementing parallel meta-heuristic algorithms and make experiments with different digital systems.
- The automatic generation of coverage models based on the use of different coverage metrics.

7.1 Contributions

An improved methodology in order to generate optimal functional coverage in digital systems.
A new schema in order to verify the functionality of digital systems based on a hybrid method (combination of meta-heuristic and dynamic methods) which allow outperforming the original designs based on the verification.
A hybrid method which performs the Directed Test Vector Generation based on compact meta-heuristic algorithms, coverage models and cost functions.
The application for the first time of compact meta-heuristics algorithms to the Directed Functional Verification.

7.1.1 Published papers in Journals

- Alfonso Martinez Cruz, Ricardo Barrón Fernández, Herón Molina Lozano, Marco Antonio Ramírez Salinas and Luis Alfonso Villa Vargas. **Automated Functional Test Generation for Digital Systems Through a Compact Binary Differential Evolution Algorithm.** Journal of Electronic Testing. Vol. 31, Num 4. pp 361 - 380. September 2015. DOI: 10.1007/s10836-015-5540-6

7.1.2 Published papers in conferences

- Alfonso Martínez Cruz, Ricardo Barrón Fernández and Herón Molina Lozano. **Automated Functional Coverage for a Digital System Based on a Binary Differential Evolution Algorithm.** 1st. BRICS Countries Congress (BRICS-CCI), Recife, Porto de Galinhas beach, Brazil, September 2013.
- Alfonso Martínez Cruz, Ricardo Barrón Fernández and Herón Molina Lozano. **Automated Functional Coverage for a Digital System Based on Particle Swarm Optimization algorithm.**. Design Automation Conference (DAC), San Francisco, CA , June 1-5, 2014.
- Alfonso Martínez Cruz, Ricardo Barrón Fernández and Herón Molina Lozano. **Automated functional coverage directed for complex digital systems.** Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on. Playa del Carmen, 6-8 Oct. 2014. pp 155 - 156

- Alfonso Martínez Cruz, Ricardo Barrón Fernández and Herón Molina Lozano. **Automated Functional Coverage Model for Digital Systems.** CORE 2014, 14o Congreso Internacional en Ciencias de la Computación. Mexico, Distrito Federal, 12-14 November, 2014.

Glossary

Application Specific Integrated Circuit. It is an customized integrated circuit for a particular application. In other words that is intended to be used for particular use.
Circuito integrado de aplicación específica (ASIC). Es un circuito integrado personalizado para una aplicación particular, en otras palabras que se pretende utilizar para uso particular.
Aptitude. Is a assigned value to each individual and indicates how good it is compared to others.
Aptitud. Valor que se asigna a cada individuo y que indica que tan bueno es respecto a los demás.
Artificial intelligence. A name for a paradigm in which people attempt to elicit intelligence from machines.
Inteligencia Artificial.Nombre de un paradigma en el cual la gente pretende generar inteligencia desde las maquinas.
Assertion. A given property that is expected to hold within a specific design.
Afirmación. Es una propiedad dada que se espera que se cumpla en un diseño especiíco.
Attribute. In the context of a device, it is a parameter or characteristic of an input or output of an interface. In the context of coverage model it is a parameter, or dimension of the model.
Atributo. En el contexto de un dispositivo, es un parámetro o característica de una entrada o salida de una interface. En el contexto de modelo de cobertura, un parámetro o dimensión del modelo.
Automatic Test Pattern Generation (ATPG) It is an automatic method of electronic design used to search an input sequence, that it is applied to a digital circuit to distinguish between correct operation of the circuit and a bad operation.
Generación automática de patrones de prueba. Es un método automático de diseño electrónico utilizado para buscar una secuencia de entrada, que cuando se aplique a un circuito digital

permita distinguir entre un correcto funcionamiento del circuito y un funcionamiento con errores.

Bite. Is the minimum unit of information. They may be taking the values 1 or 0.

Bit. Es la unidad mínima de información. Que puede ser tomar los valores de 1 o 0.

Complex digital system. It is one circuit composed of a large number of semiconductor components that could control numerous devices , from computers to mobile phones and microwave ovens.

Circuito digital complejo. Es aquel circuito compuesto de un gran número de componentes semiconductores que podría controlar numerosos aparatos, desde computadoras hasta teléfonos móviles y hornos microondas.

Coverage. Es la medida que especifica en que proporción se ha realizado la verificación de un sistema digital.

Cobertura. It is the measure specified in that proportion has carried out the verification of a digital system.

Coverage metric It is an attribute that is used as a unit of measure and is remembered, which defines a dimension of space coverage.

Métrica de cobertura. Es un atributo que es utilizado como una unidad de medida y es recordado, el cual define una dimensión del espacio de cobertura.

Coverage property. A property that, when true, indicates that a condition of interest has occurred. The occurrence is remembered in a database for further analysis.

Propiedad de cobertura. Una propiedad que, cuando es cierta, indica que una condición de interés ha ocurrido. La ocurrencia es recordada en una base de datos para análisis posteriores.

Coverage model. An abstract representation of the device behavior consisting of attributes and their inter-relationships.

Modelo de cobertura. Una representación abstracta del comportamiento de un dispositivo compuesto de atributos y sus inter-relaciones.

Coverage report. It summarizes the progress of verification statements, as a measure of coverage, capturing all facets of coverage at multiple levels of abstraction.

Reporte de cobertura. Es un resumen del progreso de verificación de estados, como una medida de cobertura, capturando todas las facetas de cobertura en múltiples niveles de abstracción.

Coverage space. It is a multi-dimensional region of the coverage attributes and their values.

Espacio de cobertura. Es una región multi-dimensional de los atributos de cobertura y sus valores.

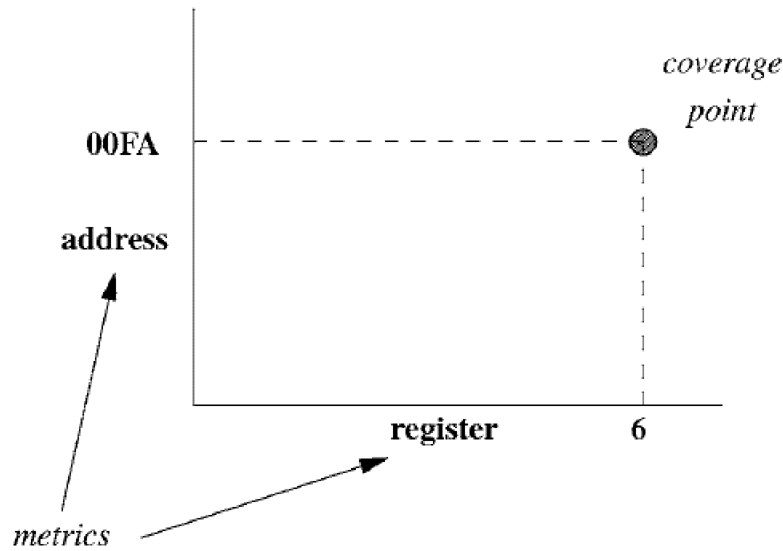


Figure 7.1. Coverage space example.

Controllability. Is the level at which the state changes in the internal nodes of the circuit can be controlled when changes occur at the inputs of the device.
Controlabilidad. Es el nivel en el cual se pueden controlar los cambios de estado en los nodos internos del circuito cuando se generan cambios en las entradas.

Coverage based on verification. It is a methodology where coverage planning precedes the rest of the verification process. Planning coverage is defined defining a strategy to measure verification progress. For example using code coverage and strategies that could help to increase the coverage.
Cobertura basada en la verificación. Es una metodología en la cual, la planificación de la cobertura precede el resto del proceso de verificación. La planificación de la cobertura se plantea definiendo una estrategia para medir el progreso de la verificación, por ejemplo utilizando cobertura de código y tácticas que podrían ayudar a incrementar la cobertura.

Corner cases. Design verification scenario which is difficult or not commonly covered in the test.
Casos corner. Escenario de verificación de un diseño que es difícil o no comúnmente cubierto en la prueba.

Constraint. A condition (usually on the input signals) which limits the set of behavior to be

considered during verification. A constraint may represent real requirements (e.g., clocking requirements) on the environment in which the design is used, or it may represent artificial limitations (e.g., mode settings) imposed in order to partition the verification.
Restricción. Una condición (usualmente en las señales de entrada) la cual, límita el conjunto de comportamiento a ser considerado durante la verificación. Una restricción puede representar requerimientos reales (por ejemplo, requerimientos de la señal de reloj) en en entorno en el cual el diseño es utilizado, o puede representar limitaciones (modo de configuración) impuesto para llevar a cabo la partición de la verificación.

Chromosome. It is a data structure containing a string of design parameters or genes. This data structure can be stored e.g. as a bit string or an array of integers.
Cromosoma. Es una estructura de datos que contiene una cadena de parametros de diseño o genes. Esta estructura de datos puede almacenarse, por ejemplo, como una cadena de bits o un arreglo de enteros.

Crossover. Operator that builds a new chromosome combining parts of each of the parent chromosomes.
Cruza. Operador que forma un nuevo cromosoma combinando partes de cada uno de los cromosomas padres.

Digital system. It is a set of devices for the generation , transmission, processing or storage of digital signals. Also a digital system is a combination of devices designed to manipulate physical quantities or information that are represented in digital form; that is, they can only take discrete values.
Sistema digital. Es un conjunto de dispositivos destinados a la generación, transmisión, procesamiento o almacenamiento de señales digitales. También un sistema digital es una combinación de dispositivos diseñados para manipular cantidades físicas o información que estén representadas en forma digital; es decir, que sólo puedan tomar valores discretos.

Device Under Test (DUT). Device to be checked. This device is distinguished from device under verification (DUV) in that this is verified as a DUT is tested device.
Dispositivo bajo prueba. Dispositivo a ser verificado. Este dispositivo es distinguido del dispositvo bajo verificación (DUV) en que este ultimo es verificado mientras un dispositivo DUT es probado.

Device Under Verification (DUV). It refers to the device to be checked. it means where the functional verification process applies.
Dispositivo bajo verificación. Se refiere al dispositivo a ser verificado. Sobre el cual se aplica el

proceso de verificación funcional.

Elitism. Mechanism that ensures that the chromosomes of the better members of a population will pass to the next generation without being altered by genetic operators.

Elitismo. Mecanismo utilizado para asegurar que los cromosomas de los miembros mas aptos de una población se pasen a la siguiente generación sin ser alterados por los operadores geneticos.

Elitism strategy. In a genetic algorithm, ensuring that the individual chromosome with the highest fitness is always copied into the next generation.

Estrategia de elitismo. En un algoritmo genetico, asegura que el cromosoma del individuo con mayor valor de aptitud es copiado siempre en la siguiente generación.

Evolutionary computation. Encompasses methods of simulating evolution on a computer. The field includes research in genetic algorithms, evolution strategies, evolutionary programming, genetic programming, particle swarm optimization, artificial life, etc.

Computo evolutivo. Abarca los metódos de simulación de evoluación en una computadora. El campo incluye investigación en algoritmos genéticos, algoritmo de optimización por cúmulo de partículas, vida artificial y otras.

Fault. *Falla.* A deviation of a specified behavior system.

Falla. Es una desviación del comportamiento especifico (correcto) del sistema.

Functional coverage. Is a coverage whose metric is derived from a functional specification or design. A collection of user-defined metrics if all required settings have been tested.

Cobertura funcional. Es aquella cobertura cuya métrica es derivada de una especificación funcional o de diseño. Una colección de métricas definidas por el usuario si todos los escenarios requeridos han sido probados.

Formal verification. A mathematical comparison against an implementation specification or requirement to determine whether implementation can violate this specification.

Verificación Formal. Una comparación matemática de una implementación againts una especificación o requerimiento para determinar si la implementación puede violar esta especificación.

Finit State Machine (FSM) It is a machine, which can be completely described by a finite set of states, being in a state at some point over a set of rules which determine, when moving from one state to another.

Maquina de estados finitos. Es una máquina, la cual, puede ser totalmente descrita por un conjunto finito de estados, estando en un estado en algún momento, más un conjunto de reglas, las cuales, determinan, cuando se mueve de un estado a otro.

Formal verification. It means to the use of mathematical modeling and analysis of the functional verification of the design behavior.
Verificación formal. Se refiere al uso de modelos matemáticos y el análisis de la verificación funcional del comportamiento del diseño.

Gene. It is a subsection of the chromosome (usually) encodes the parameter value of a single chromosome.
Gen.Es una subseccion del cromosoma que (usualmente) codifica el valor de un solo parametro del cromosoma.

Genotype. Coding (e.g binary) of the parameters that represent a solution to the problem to be solved.
Genotipo. Codificación (por ejemplo binaria) de los parametros que representan una solución del problema a resolver.

Generation. Each of the aptitude measures and the creation of a new population through reproduction operators.
Generación. Cada una de las medidas de aptitude y la creación de una nueva población por medio de operadores de producción.

Hardware verification. It the proof that a circuit or system (implementation) behaves according to a given set of requirements. Is a process which checks the correct operation of a design.
Verificación de Hardware. Es la prueba de que un circuito o un sistema (la implementación) se comportan acorde a un conjunto dado de requerimientos. Es decir es un proceso mediante el cual se demuestra el funcionamiento correcto de un diseño.

Hybrid verification. It means to the integration of verification technologies into a unified platform to create a higher level of automation.
Verificación híbrida. Se refiere a la integración de tecnologías de verificación en una plataforma unificada para crear un nivel más alto de automatización.

Hardware Description Language (VHSIC, VHDL, VHSIC). Is a hardware description language which is used in an automatic electronic design to describe digital and mixed systems such as gate arrays and programmable logic integrated circuits. This can be done at different abstraction levels. Examples include VHDL, Verilog, SystemVerilog, SystemC.
Lenguaje de descripción de hardware. Son lenguajes de programación utilizados para para describir sistemas digitales y mixtos como son arreglos de compuertas lógicos programables y circuitos integrados.Esto puede hacerse en distintos niveles de abstracción. Algunos ejemplos

son VHDL, Verilog, SystemVerilog, SystemC.

Model. It is an abstraction or approach the behavior of a logic design (digital system).

Modelo. Es una abstracción o aproximación del comportamiento de un diseño lógico (sistema digital).

Mutation. Operator which builds a new chromosome through changes (usually small of the values of genes of one father chromosome.

Mutación. Operador que forma un nuevo cromosoma a traves de alteraciones (usualmente pequeñas) de los valores de los genes de un solo cromosoma padre.)

Observabillity. Is the analysis of output values obtained according to the circuit performance exercise.

Observabilidad. Es el análisis de los valores de salida obtenidos acorde al desempeño que ejercitamos del circuito.

Optimization. The adjustment of a system in order to minimize or maximize the result of some function.

Optimización. Es el ajuste de un sistema para minimizar o maximizar el resultado de alguna función.

Productivity on the verification. It is defined as the ability to control large designs in a short time. Is other words is the measure of the amount of manual effort which is involved in the project verification.

Productividad en la verificación. Es definida como la habilidad para manejar grandes diseños en un corto tiempo. Es otras palabras es la medida de la cantidad de esfuerzo manual que es involucrado en el proyecto de verificación.

Phenotype. It is the codification of the chromosome. That is, the obtained values by passing the representation (binary) to that is used for the objective function.

Fenotipo. Es la codificación del cromosoma. Es decir, los valores obtenidos al pasar de la representación (binaria) a la usada por la función objetivo.

Representation. It amounts to specifying a mapping from the set of phenotypes onto set of genotypes that said to represent them.

Representación. Equivale a especificar un mapeo del conjunto de fenotipos en los genotipos que representan.

System On a Chip. *SOC*. It is an integrated circuit that integrates all the elements of a computer. It may contain digital, analog circuits, and generally mixed signals or radio frequency functions . All contained in a single integrated circuit.

Sistema en un circuito integrado. Es un circuito integrado que integra todos los elementos de una computadora. Éste puede contener circuitos digitales, analógicos, o señales mixtas y generalmente funciones de radiofrecuencia. Todas contenidas en un solo circuito integrado.

Sintesis. It is the process of taking a written design in hardware description language compiling a list of interconnection gates, which are selected from a library provided by the user.

Síntesis. Es el proceso de tomar un diseño escrito en un lenguaje de descripción de hardware compilandolo en una lista de interconexión de compuertas, las cuales, son seleccionadas desde una biblioteca provista por el usuario de varias compuertas.

Testbench. It refers to a simulation code used to create a default entry sequence for a design, then optionally to observe the response. It is a simulation of environment which can contains design. It checks whether the RTL implementation meets the design specification or not. This environment creates conditions and unexplored invalid and valid conditions to test the design.

Testbench. Se refiere a un código de simulación utilizado para crear una predeterminada secuencia de entrada para un diseño, entonces opcionalmente para observar la respuesta. Es un simulador del entorno en el cual, el diseño podría residir. Éste revisa si la implementación RTL cumple con la especificación del diseño o no. Este entorno crea condiciones invalidas y no exploradas así como condiciones validas para probar el diseño.

Test vector. It is an n-tuple of binary values where each bit is applied to each input of the device under verification (DUV) with the goal of performing functional verification of a digital system

Vector de prueba. Es una n-tupla de valores binarios en la que cada bit es aplicado en cada entrada del dispositivo bajo verificación (DUV), con el objetivo de realizar la verificación del funcionamiento de un sistema digital.

Verification. The process showing the functionality of a design is preserved in its implementation.

Verificación. El proceso de demostrar que la funcionalidad de un diseño se preservada en su implementación.

Verification interface. A level of abstraction at which a verification process is performed if a simulation is used. This is a common interface, in which the stimuli are applied, the behavioral response is reviewed and coverage is measured.

Interfaz de verificación. Un nivel de abstracción en el cual un proceso de verificación es realizado. Si una simulación es utilizada. Esto es una interfaz común, en la cual, los estímulos son aplicados, la respuesta comportamental es revisada y se mide la cobertura.

Verification plan. It is the mechanism which ensures that the essential characteristics of a

design will be properly verified. It describes the scope of the verification problem for the device and serves as a functional specification for the verification environment. In the verification plan design parts to be checked are identified and the way how they will be verified.
Plan de verificación. Es el mecanismo mediante el cual se asegura que las características esenciales de un diseño sean verificadas apropiadamente. Éste describe el alcance del problema de verificación para el dispositivo y sirve como especificación funcional para el entorno de verificación. En el plan de verificación se identifican las partes del diseño que se deben verificar y la forma en que deben verificarse.

Bibliography

[1] Andrew Piziali. *Functional Verification Coverage Measurement and Analysis*. 2008.

[2] Irina Rancea and Valentin Sgarciu. Functional verification of digital circuits using a software system. *Proc. IEEE International Conference on Automation*, 1(1):152–157, May 2008.

[3] Young-Jin Oh and Gi-Yong Song. Simple hardware verification platform using systemverilog. *Proc. IEEE TENCON*, 1(1):pp 1414 – 1417, November 2011.

[4] Salim Kalla Riadh Hocine, Hamoudi Kalla and Chafik Arar. A methodology for verification of embedded systems based on systemc. *Proc. IEEE International Conference on Complex Systems (ICCS)*, 1(1):pp 1–6, November 2012.

[5] Armin Biere. Verifying sequencial behavior with model checking. *ASIC, 2001. Proceedings. 4th International Conference on*, 1(1):29 – 32, October 2001.

[6] M. Fújital Jawahar Jain, Amit Narayan and A. Sangiovanni-Vincentelli. A survey of techniques for formal verification of combinational circuits. *IEEE Computer Design: VLSI in Computers and Processors*, 1(1):pp 445–454, October 1997.

[7] Serdar Tasiran and Kurt Keutzer. Coverage metrics for functional validation of hardware designs. *IEEE Design & Test of Computers.*, 18(4):pp 36–45, August 2001.

[8] JohnR. Wallack and Ramaswami Dandapani. Coverage metrics for functional tests. *Proc. IEEE VLSI Test Symposium*, 1(1):pp 25–28, April 1994.

[9] Ulrich Heinkel Dietmar Müller Vasco Jerinic, Jan Langer. New methods and coverage metrics for functional verification. *Proc. IEEE Design, Automation and Test in Europe (DATE)*, 1(1):pp 1–6, March 2006.

[10] Orna Kupferman Hana Chockler and Moshe Vardi. Coverage metrics for formal verification. *International Journal on Software Tools for Technology*, 8(1):pp 373 – 386, August 2006.

[11] Franco Fummi Xiaoming Yu, Alessandro Fin and Elizabeth M. Rudnick. A genetic testing framework for digital integrated circuits. *International Conference on tools with Artificial Intelligence*, 1(1):1–6, November 2002.

[12] Giovanni Squillero A Manzone Fulvio Corno, Matteo Sonza Reorda and Alessandro Pincetti. Automatic test bench generation for validation of rtl-level descriptions: an industrial experience. *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE)*, 1(1):pp 385 – 389, March 2000.

[13] Elizabeth M. Rudnick Todd Dukes Mrinal Bose, Jongshin Shin and Magdy Abadir. A genetic approach to automatic bias generation for biased random instruction generation. *Evolutionary Computation*, 2(1):442–448, May 2001.

[14] Sofiene Tahar Amer Samarah, Ali Habibi and Nawwaf Kharma. Automated coverage directed test generation using a cell-based genetic algorithm. *High-Level Design Validation and Test Workshop*, 1(1):19–26, November 2006.

[15] Yunji Chen Bowen Chen Haihua Shen, Wenli Wei and Qi Guo. Coverage directed test generation: Godson experience. *Asian Test Symposium*, 1(1):321–326, November 2008.

[16] Min Li and Michael S. Hsiao. An ant colony optimization technique for abstraction-guided state justification. *Proc. IEEE International Test Conference*, 1(1):pp 1–10, November 2009.

[17] Kelson Gent and Michael S. Hsiao. Functional test generation at the rtl using swarm intelligence and bounded model checking. *Proc. IEEE 22nd Asian Test Symposium*, 1(1):pp 233–238., November 2013.

[18] Godfrey C. Onwubolu and Donald Davendra. Differential evolution: A handbook for global permutation-based combinatorial optimization. *Springer-Verlag Berlin Heidelberg*, 1(1):1–226, – 2009.

[19] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 1(1):pp 341–359, – 1995.

[20] Ponnuthurai Nagaratnam Suganthan Swagatam Das. Differential evolution: A survey of the state-of-the-art. ieee transactions on evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 15(1):pp 4–31, February 2011.

[21] Ricardo Barron Fernandez Alfonso Martinez Cruz and Heron Molina Lozano. Automated functional coverage for a digital system based on a binary differential evolution algorithm. *Proc. IEEE Congress on Computational Intelligence & 11th Brazilian Congress on Computational Intelligence (BRICS)*, 1(1):pp 92 – 97, September 2013.

[22] Shai Fine and Avi Ziv. Coverage directed test generation for functional verification using bayesian networks. *Design Automation Conference*, 15(5):286–291, June 2003.

[23] Valeria Bertacco Ilya Wagner and Todd Austin. Microprocessor verification via feedback-adjusted markov models. *Computer-Aided Design of Integrated Circuits and Systems*, 26(6):1126–1138, June 2007.

[24] Tao Lv Tiancheng Wang Xiaowei Li Jian Wang, Huawei Li and Sandip Kundu. Abstraction-guided simulation using markov analysis for functional verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1(1):1–13, March 2015.

[25] Gustavo Neuberger Jorge Tonfat and Ricardo Reis. Functional verification of logic modules for a gigabit ethernet switch. *Test Workshop (LATW)*, 1(1):1–4, March 2011.

[26] Yannick Zoccarato Patrice Vado, Yvon Savaria and Chantal Robach. A methodoly for validating digital circuits with mutation testing. *International symposium on circuits ans systems*, 5(1):343 – 346, May 2000.

[27] Anton Chepurov Hanno Hantson Raimund Ubar Jaan Raik, Urmas Repinski and Maksim Jenihhin. Automated design error debug using high-level decision diagrams and mutation operators. *Microprocessors and Microsystems*, 37(1):505 – 513, December 2012.

[28] Vincent Beroulle Youssef Serrestou and Chantal Robach. Functional verification of rtl designs driven by mutation testing metrics. *Digital System Design Architectures, Methods and Tools*, 1(1):pp 222 – 227, August 2007.

[29] Xiaoke Qin and Prabhat Mishra. Efficient directed test generation for validation of multicore architectures. *Quality Electronic Design (ISQED)*, 1(1):1–8, March 2011.

[30] Wolfgang Mueller Tao Xie and Florian Letombe. Mutation-analysis driven functional verification of a soft microprocessor. *SOC Conference (SOCC)*, 1(1):283–288, September 2012.

[31] David Sheridan Samuel Hertz and Shobha Vasudevan. Mining hardware assertions with guidance from static analysis. *Computer-Aided Design of Integrated Circuits and Systems*, 32(6):952–965, June 2013.

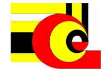
[32] Jay Bhadra When Chen, Li-Chung Wang and Magdy Abadir. Simulation knowledge extraction and reuse in constrained random processor verification. *Design Automation Conference (DAC)*, 1(1):1–6, June 2013.

[33] Sanjay Patel David Tcheng Bill Tuohy Shobha Vasudevan, David Sheridan and Daniel Johnson. Goldmine: Automatic assertion generation using data mining and static analysis. *Proc. IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, 1(1):pp 626 – 629, March 2010.

[34] Edgar Leonardo Romero & Marius Strum and Wang Jiang Chau. Manipulation of training sets for improving data mining coverage driven verification. *Journal of Electronic Testing, Theory and Applications (JETTA)*, 29(2):pp 223–236, March 2013.

[35] Sanjay Patel David Tcheng Bill Tuohy Shobha Vasudevan, David Sheridan and Daniel Johnson. Goldmine: Automatic assertion generation using data mining and static analysis. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1(1):1530–1591, March 2010.

[36] Serdar Tasiran and Kurt Keutzer. Coverage metrics for functional validation of hardware designs. *Design & Test of Computers*, 18(4):36–45, Jul-Aug 2001.



[37] Franco Fummi Iuseppe Di Guglielmo, Luigi Di Guglielmo and Graziano Pravadelli. Efficient generation of stimuli for functional verification by backjumping across extended fsms. *Electron Test, Springer*, 1(27):137–162, March 2011.

[38] Robert B. Jones Mark D. Aagaard and Carl-Johan H. Seger. Combining theorem proving and trajectory evaluation in an industrial environment. *Design Automation Conference*, ISBN:0-89791-964-5(1):538–541, June 1998.

[39] Roope Kaivola, Aagaard, and MarkD. Divider circuit verification with model checking and theorem proving. *Lecture Notes in Computer Science: Theorem Proving in Higher Order Logics*, 1869(1):338–355, 2000.

[40] J. O'Leary, R. Kaivola, and T. Melham. Relational ste and theorem proving for formal verification of industrial circuit designs. *Formal Methods in Computer-Aided Design (FMCAD)*, 1(1):97–104, October 2013.

[41] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, and A. Ziv. Genesys-pro: innovations in test program generation for functional processor verification. *Design Test of Computers, IEEE*, 21(2):84–93, March 2004.

[42] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Lecture Notes in Computer Science*, 131(1):52 – 71, – 1982.

[43] Carl-Johan Seger. An introduction to formal hardware verification. *IEEE,*, 1(1):1–27, – 2001.

[44] Armin Biere Mukul R. Prasad and Aarti Gupta. A survey of recent advances in sat-based formal verification. *Software Tools for Tehcnology Transfer manuscript, IEEE*, 1(1):1 –18, 2002.

[45] Jayanand Asok Kumar and Shobba Vasudevan. Verifying dynamic power management schemes using statistical model checking. *Design Automation Conference (ASP-DAC)*, 12(7):579 – 584, January 2012.

[46] Ali Alphan Bayazit and Sharad Malik. Complementary use of runtime validation and model checking. *Computer-Aided Design*, 1(1):1052–1059, November 2005.

[47] Jayanand Asok Kumar and Shobha Vasudevan. Verifying dynamic power management schemes using statistical model checking. *Proc. IEEE Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, 1(1):pp 574–584, January 2012.

[48] Cristina Tudose Raluca Lefticaru, Florentin Ipate. Automated model design using genetic algorithms and model checking. *Proc. IEEE Fourth Balkan Conference in Informatics*, 1(1):pp 79–84, September 2009.

[49] Ali Alphan Bayazit and Sharad Malik. Complementary use of runtime validation and model checking. *Proc. IEEE Computer-Aided Design, (ICCAD-2005)*, November(2005):pp 1049–1056, 1 1.

[50] Ulrich Kühne Daniel Große and Rolf Drechsler. Analyzing functional coverage in bounded model checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):pp 1305–1314, July 2008.

[51] Franco Fummi Andrea Fedeli and Granziano Pravadelli. Properties incompleteness evaluation by functional verification. *IEEE Transactions on Computers*, 56(4):pp 528–544, April 2007.

[52] Mingsong Chen and Prabhat Mishra. Decision ordering based property decomposition for funtional test generation. *Proc. IEEE Design, Automation & Test in Europe (DATE)*, 1(1):pp 1–6, March 2011.

[53] Thomas A. Henzinger. Symbolic model checking for real-time systems. *Logic in Computer Science*, 1(1):pp 394–406, June 1992.

[54] Shmuel Ur Oded Lachish, Eitan Marcus and Avi Ziv. Hole analysis for functional coverage data. *Design Automation Conference*, 39(1):807–812, June 2002.

[55] Yi-Shing Jian Kang, Sharad C. Seth and Vijay Gangaram. Efficient selection of observation points for functional tests. *Proc. IEEE 9th International Symposium on Quality electronic Design*, 1(1):pp 236–241, March 2008.

[56] Shmuel Ur Oded Lachish, Eitan Marcus and Avi Ziv. Hole analysis for functional coverage data. *Proc. IEEE Design Automation Conference (DAC)*, 1(1):pp 807–812, June 2002.

[57] Adam C. Krolnik Harry D. Foster and David J. Lacey. *Assertion based Device*. 2005.

[58] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research: Applications of Integer Programming*, 13(5):533–549, – 1986.

[59] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. 2007.

[60] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. *Sixth International Symposium on Micro Machine and Human Science*, 1(1):39–43, October 1995.

[61] Rainer Storn and Kenneth Price. Minimizing the real functions of the icec'96 contest by differential evolution. *Evolutionary Computation*, 1(1):pp 842–844, May 1996.

[62] A.P. Engelbrecht and G. Pampará. Binary differential evolution algorithm. *Evolutionary Computation*, 1(1):pp 1873–1879, July 2006.

[63] Xingshi He and Lin Han. A novel binary differential evolution algorithm based on artificial immune system. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, 1(1):pp 2267–2272, September 2007.

[64] Andries Engelbrecht and Gary Pampara. Binary differential evolution strategies. *Evolutionary Computation, 2007. CEC 2007*, 1(1):1942–1947, September 2007.

[65] Yanling Yang-Hu Peng Changshou Deng, Bingyan Zhao and Qiming Wei. Novel binary encoding differential evolution algorithm. *Second International Conference, ICSI 2011, Chongqing, China*, 1(1):416–423, June 2011.

[66] Thomas Weise Changshou Deng and Bingyan Zhao. Pseudo binary differential evolution algorithm. *Journal of Computational Information Systems*, 8(6):2425–2436, March 2012.

[67] Tao Gong and Andrew L. Tuson. Differential evolution for binary encoding. *Soft Computing in Industrial Applications*, 39(1):251–262, March 2007.

[68] Muhammad Ilyas Menhas Ling Wang, Xiping Fu and Minrui Fei. A modified binary differential evolution algorithm. *LSMS/ICSEE 2010*, 1(1):49–57, – 2010.

[69] Fernando G. Lobo Georges R. Harik and David E. Goldberg. The compact genetic algorithm. evolutionary computation. *IEEE Transactions*, 3(4):pp 842–844, May 1996.

[70] Chang Wook Ahn and R. S. Ramakrishna. Elitism-based compact genetic algorithms. *Evolutionary Computation, IEEE Transactions*, 7(4):pp 367–385, August 2003.

[71] Francesco Cupertino Ernesto Mininno, Ferrante Neri and David Naso. Compact differential evolution. *Evolutionary Computation*, 15(1):pp 32–54, December 2010.

[72] Ferrante Neri and Ernesto Mininno. Memetic compact differential evolution for cartesian robot control. *IEEE Computational Intelligence Magazine.*, 5(2):pp 54–65, May 2010.

[73] George Harik and Erick Cantu Paz. The gamblers ruin problem, genetic algorithms, and sizing of populations. *Evolutionary Computation*, ISBN:0-7803-3949-5(1):7–12, April 1997.

[74] Heron Molina Lozano Marco Ramirez Salinas Alfonso Martinez Cruz, Ricardo Barron Fernandez and Luis Alfonso Villas. Automated functional test generation for digital systems through a compact binary differential evolution algorithm. *Journal of Electronic Testing*, 31(4):pp 361 – 380, September 2015.

[75] Vecchi Mario P and Scott Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(4):pp 215–222, October 1983.

[76] Yannick Zoccarato Patrice Vado, Yvon Savaria and Chantal Robach. A methodology for validating digital circuits with mutation testing. *Circuits and Systems*, 1(1):343–346, June 2000.

[77] Russell Eberhart and James Kennedy. A discrete binary version of the particle swarm algorithm. *Sixth International Symposium on Micro Machine and Human Science*, 1(1):39–43, October 1995.

[78] Design Automation Standards Committee. Design automation standards committee 1800-2009 - ieee standard for systemverilog–unified hardware design, specification, and verification language. *IEEE Standards Association Corporate Advisory Group.*, 1(1):1–1285, November 2009.

[79] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 1(1):341–359, March 1995.

[80] Serdar Tasiran and Kurt Keutzer. Coverage metrics for functional validation of hardware designs. *Design & Test of Computers, IEEE*, 18(4):36–45, July/August 2001.

[81] Alair Dias Junior and Diogenes Cecilio da Silva Junior. Code-coverage based test vector generation for systemc designs. *VLSI, 2007. ISVLSI '07*, 1(1):198–206, March 2007.

[82] Charalambos Ioannides and Kerstin I. Eder. Coverage-directed test generation automated by machine learning - a review. *ACM Transactions on Design Automation of Electronic Systems*, 17(1):7–21, January 2012.

[83] V. Subedha and S. Sridhar. An efficient coverage driven functional verification system based on genetic algorithm. *European Journal of Scientific Research*, 81(4):533–542, 2012.

[84] Marius Strum Carlos Iván Castro Márquez, Edgar Leonardo Romero Tobar and Wang Jiang Chau. A functional verification methodology based on parameter domains for efficient input stimuli generation and coverage modeling. *Journal of Electronic Testing*, 27(4):485–503, August 2011.

[85] Matteo Sonza Reorda Fulvio Corno, Ernesto Sánchez and Giovanni Squillero. Automatic test program generation: A case study. *Design & Test of Computers, IEEE*, 21(2):102–109, 2004.

[86] Yi-Shing Chang Jian Kang, Sharad C. Seth and Vijay Gangaram. Efficient selection of observation points for functional tests. *9th International Symposium on Quality Electronic Design*, 1(1):236–241, 2008.

[87] James Kennedy and Russell C. Eberhart. A discrete binary version of the particle swarm algorithm. *Computational Cybernetics and Simulation*, 5(1):4104–4108, October 1997.

[88] Shaz Qadee and Serdar Tasiran. Promising directions in hardware design verification. *Proceedings of the International Symposium on Quality Electronic Design*, 1(1):1–6, March 2002.

[89] Magdy S. Abadir Jayanta Bhadra and Sandip Ray. A survey of hybrid techniques for functional verification. *Test of Computers- Advances in Functional Validation through Hybrid Techniques*, 24(2):112–122, March 2007.

[90] Armin Biere Mukul R Prasad and Aarti Gupta. A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173, April 2005.

[91] Jacob A. Abraham Dinos Moundanos and Yatin V. Hoskote. Abstraction techniques for validation coverage analysis and test generation. *IEEE TRANSACTIONS ON COMPUTERS*, 47(1):2–14, January 1998.

[92] Farn Wang. Formal verification of timed systems: a survey and perspective. *Proceedings of the IEEE*, 92(8):1283–1305, August 2004.

[93] David Lee and Mihalis Yannakakis. Principles of methods of testing finite state machines- a survey. *Proceedings of the IEEE*, 84(8):1–6, August 1996.

[94] Gal Katz and Doron Peled. Model checking-based genetic programming with an application to mutual exclusion. *Tools and Algorithms for the Construction and Analysis of Systems*, 4963(1):141–156, Jun 2008.

[95] Robert B. Jones John O'Leary W. Tom Melham D. Mark Aagaard Clark Barrett Johan Carl, Seger H. and Don Syme. An industrially effective envieronment for formal hardware verification. *Transactions on computer-aided design of integrated circuits and systems*, 24(8):1381–1405, September 2005.

[96] Laurent Fournier Eitan Marcus Michael Rimon Michael Vinov Allon Adir, Eli Almong and Avi Ziv. Genesys-pro: Innovations in test program generation for functional processor verification. *IEEE Design & Test of Computers*, 21(2):84–93, March-April 2004.

[97] Cristina Tudose Raluca Lefticaru, Florentin Ipate. Automatic model design using genetic algorithms and model checking. *IEEE, fourth Balkan Conference in Informatics*, 2(1):79–84, – 2009.

[98] Mihaela Radu. Extensive coverage of functional verification of hardware designs. *Microelectronic Systems Education, 2007. MSE '07*, 1(1):101 – 102, June 2007.

[99] Gang Chen. A short historical survey of functional hardware languages. *International Scholarly Research Network ISRN Electronics*, 2012(271836):1–11, February 2012.

[100] Alessandro Fin and Franco Fummi. Genetic algorithms: the philosopher's stone or an effective solution for high-level tpg?*. *High-Level Design Validation and Test Workshop, 2003. Eighth IEEE International*, 1(1):163 – 168, November 2003.

[101] Sharad C. Seth Jian Kang and Vijay Gangaram. Efficient rtl coverage metric for functional test selection. *25th IEEE VLSI Test Symmposium (VTS'07)*, 1(1):318–324, May 2007.

[102] Ali Habibi and Sofiene Tahar. A survey: Systemonachip design and verification. *Technical Report, Montreal, Quebec, Canada*, 1(1):1 – 32, January 2003.

[103] Laurent Arditi and Gael Clave. A semi-formal methodology for the functional validation of an industrial dsp system. *ISCAS 2000 IEEE International Symposium on Circuits and Systems*, 1(4):205–208, May 2000.

[104] Orna Kupferman Hana Chockler and Moshe Vardi. Coverage metrics for formal verification. *International Journal on software tools for technology*, 2(1):376 – 386, April 2006.

[105] Byoungju Choi Ahyoung Sung and Jangsoo Lee. Interaction mutation testing. *Copyright 2003 Chillarege Press*, 2(1):1–2, – 2003.

[106] Eamonn Otoole Eamonn Linehan and Siobhan Clarke. Model-driven automation for simulation-based functional verification. *ACM Transactions on Design Automation of Electronic System*, 17(3):31–56, June 2002.

[107] Kurt Shultz Jun Yuan and Carl Pixley. automatic vector generation using constraints and biasing. *Journal of electronic Testing: Theory and Applications*, 16(1):107–120, July 2000.

[108] L. Fanucci S. Saponara and M. Coppola. Design and coverage-driven verification of a novel network-interface ip macrocell for network-on-chip interconnects. *Microprocessors and Microsystems*, 35(1):579–592, July 2011.

[109] A. S. Kamkin and M. M. Chupilko. Survey of modern technologies of simulation-based verification of hardware. *Programming and Computer Software*, 37(3):147–152, September 2011.

[110] Eamonn Linehan and Siobhán Clarke. An aspect-oriented, model-driven approach to functional hardware verification. *Journal of Systems Architecture*, 58(1):195–208, February 2012.

[111] Marius Strum Edgar Leonardo Romero and Wang Jiang Chau. Manipulation of training sets for improving data mining coverage-driven verification. *J Electron Test*, 29(1):223–236, March 2013.

[112] Sasan Iman and Sunita Joshi. *The e Hardware Verification Languaje*. Springer Science-Kluwer Academic Publishers, United States of America, 1 edition, 2004.

[113] Chris Spear and Greg Tumbush. *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. 2012.

[114] Carl Pixley Jun Yuan and Adnan Aziz. *CONSTRAINT-BASED VERIFICATION*. 2006.

[115] Hamilton B. Carter and Shankar Hemmady. *Metric Driven Design Verification:An Engineers and Executives Guide to First Pass Success*. 2007.

[116] Rainer M. Storn Kenneth V. Price and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. 2005.

[117] James Kennedy and Russell C. Eberhart with Yuhui Shi. *Swarm Intelligence*. Academic Press, first edition edition, 2001.

[118] John W. O'Leary Tom Melham Mark D. Aagaard Clark Barrett Carl Johan H. Seger, Robert B. Jones and Don Syme. An industrially effective environment for formal hardware verification. *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, 24(9):1381–1405, September 2005.