PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

FUNCTIONAL VERIFICATION FRAMEWORK OF AN AES ENCRYPTION MODULE

Thesis theme to opt for the electronic engineer's title

Frank Pedro Plasencia Balabarca

ADVISORS: MSc. Edward Máximo Mitacc Meza

MSc. Mario Andrés Raffo Jara

Lima, 2017

To God who has granted me with intelligence and comprehension and has blessed me so much.

To my mother Mirtha who has been always by my side giving me her support and love.

To my uncle Franklin who has been as a mentor since I remember.

To the memory of my father Victor.

To my soul-mate Rosmery who has inspired me in many ways since the day I met her.

To my advisors and friends Mario and Edward who have wisely guided me through this research and have honored me with their honest friendship.

To my friends of GuE PUCP Marco, Jair, Carlo, Hector, Haris and Erick with whom I shared countless hours and nice moments at the Microelectronics laboratory.

To my best friends in life Claudia, Jose, Héctor and Patricia who have always been there for me and which I consider as my family.

The only way to do great work is to love what you do.
if you haven't found it yet, keep looking.
Don't settle.
As with all matters of the heart, you'll know when you find it.
Steve jobs

Das wahre Zeichen von Intelligenz ist nicht das Wissen,
sondern die Vorstellungskraft.
Albert Einstein

Abstract

Over the time, the development of the digital design has increased dramatically and nowadays many different circuits and systems are designed for multiple purposes in short time lapses. However, this development has not been based only in the enhancement of the design tools, but also in the improvement of the verification tools, due to the outstanding role of the verification process that certifies the adequate performance and the fulfillment of the requirements. In the verification industry, robust methodologies such as the Universal Verification Methodology (UVM) are used, an example of this is [1], but they have not been implemented yet in countries such as Peru and they seem inconvenient for educational purposes.

This research propose an alternative methodology for the verification process of designs at the industry scale with a modular structure that contributes to the development of more complex and elaborated designs in countries with little or none verification background and limited verification tools. This methodology is a functional verification methodology described in SystemVerilog and its effectiveness is evaluated in the verification of an AES (Advance Encryption Standard) encryption module obtained from [2]. The verification framework is based on a verification plan (developed in this research as well) with high quality standards as it is defined in the industry. This verification plan evaluates synchronization, data validity, signal stability, signal timing and behavior consistency using Assertions, functional coverage and code coverage.

An analysis of the outcomes obtained shows that the AES encryption module was completely verified obtaining 100% of the Assertions evaluation, 100% of functional verification and over 95% of code coverage in all approaches (fsm, block, expression, toggle). Besides, the modular structure defines the intercommunication with the Design only in the bottom most level, which facilitates the reuse of the verification framework with different bus interfaces. Nonetheless, this unit level verification framework can be easily instantiated by a system level verification facilitating the scalability. Finally, the documentation, tutorials and verification plan templates were generated successfully and are aimed to the development of future projects in the GuE PUCP (Research group in Microelectronics). In conclusion, the methodology proposed for the verification framework of the AES encryption module is in fact capable of verifying designs at the industry scale with high level of reliability, defining a very detailed and standardized verification plan and containing a suitable structure for reuse and scalability.

DEPARTAMENTO DE
INGENIERÍA
SECCIÒN ELECTRICIDAD Y ELECTRÒNICA

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DEL PERÚ

THESIS THEME TO OPT FOR THE ELECTRONIC ENGINEER'S TITLE

Title	:	Functional verification framework for an AES encryption module
Area	:	Circuits and Systems
Advisors	:	MSc. Eng. Edward Máximo Mitacc Meza
		MSc. Eng. Mario Andrés Raffo Jara
Student	:	Frank Pedro Plasencia Balabarca
Code	:	20135025
Date	:	November 10^{th}, 2017

07 DIC. 2017

1426

Description and Objectives:

The fast development of the integrated circuits has produced a progressive increment on the complexity level of these circuits which integrate more functions every time and present behaviors very hard to predict. All of these circuits' designs should go under an adequate verification process with high standards of reliability and over the time the traditionally used direct verification has presented many limitations in time efficiency and reliability. Hence, the digital design industry has developed the functional verification which has many advantages over the direct verification and its use is greatly extended over many countries. However, it has not been implemented in the national context yet, and the investigation in this research area is almost none. The main objective of this research is the development of a pioneer national functional verification framework for a design at the industry scale such as the AES (Algorithm encryption standard) encryption module. In order to guarantee the scalability and reusability, the object oriented HVL (hardware verification language) SystemVerilog is used. The specific objectives are:

- Implement a functional verification framework described in SystemVerilog and based on all the specifications contained in the verification plan elaborated as well in this document.
- Define a functional verification methodology that can be reproduced in any given design of similar or lower complexity.
- Elaborate a verification plan template with technical quality standards similar to the actual industrial standards so it can be employed as a reference in any future project in PUCP.
- Write documentation about the study of SystemVerilog and develop simple and concise tutorials of its application so it can be easily utilized in any verification process.

MÁXIMO 50 PÁGINAS

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

M. Sc. Ing. WILLY CARRERA SORIA
Coordinador de la Especialidad de Ingeniería Electrónica

DEPARTAMENTO DE
INGENIERÍA
SECCIÒN ELECTRICIDAD Y ELECTRÒNICA

PONTIFICIA
UNIVERSIDAD
CATÓLICA
DEL PERÚ

THESIS THEME TO OPT FOR THE ELECTRONIC ENGINEER'S TITLE

Title : Functional verification framework for an AES encryption module

Index

Introduction

1. Verification basics and problem formulation

2. A deeper approach to the functional verification and the AES encryption module

3. Elaboration and implementation of the functional verification framework

4. Simulation and result evaluation

Conclusions

Bibliography

Appendices

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA
M. Sc. Ing. WILLY CARRERA SORIA
Coordinador de la Especialidad de Ingeniería Electrónica

Contents

List of Figures

List of Tables

Introduction

The Electronics field has performed an outstanding role in the society development with several contributions such as the creation of multiple devices to fulfill people's needs and improve their living conditions. Currently, in this industry many devices are developed for different applications in short time lapses. Internally, these devices are constituted by digital systems each of which groups a great variety of integrated circuits. These circuits have been an essential factor in the electronics evolution which has remained over the time according to what Gordon Moore predicted: the density of transistors per square inch in an integrated circuit will increase yearly [6]. The current integrated circuits overpass the million of transistors per square inch, several input/output ports, and a multitasking performance in a single chip. Hence, the complexity level of numerous designs is so high that the behavior under any set of stimuli is greatly unpredictable.

In the industry, every design is submitted to an exhaustive verification process before its massive production since those circuits are normally involved in some critical areas such as medicine, information security and the stock exchange. In these areas, any failure could cause severe consequences, some of them are: money loss, information leak and death. The verification process mainly consists of a comparison between the design requirements and the actual behavior of the circuit under any possible condition. Besides, this process can widely vary among the different designs because of the functionality and the requirements.

In the communication systems, the encryption modules are in charge of guaranteeing the information integrity, protecting the information transmissions from any possible attack. Therefore, the design of those modules must be tested under very high verification standards to ensure its correctness and reliability.

This document gives a deeper perspective of the verification, specifying the types and its differences, highlighting the convenience of the functional verification employment. Furthermore, a complete functional verification framework is developed for an AES [1](Advance Encryption Standard) encryption module which is considered a design at an industry scale.

[1]The AES is described in detail in section 2.1

Chapter 1

Verification basics and problem formulation

1.1 Verification fundamentals

In practice, the verification process is done by the verification framework which generates input stimuli and monitors output behavior of the digital system that is normally known as the Design under verification (DUV). These tasks performed by the verification framework allow the accomplishment of a comparison between the design requirements and the actual behavior of a given design. Figure 1.1 illustrates the interaction enclosed by the DUV and the verification framework.

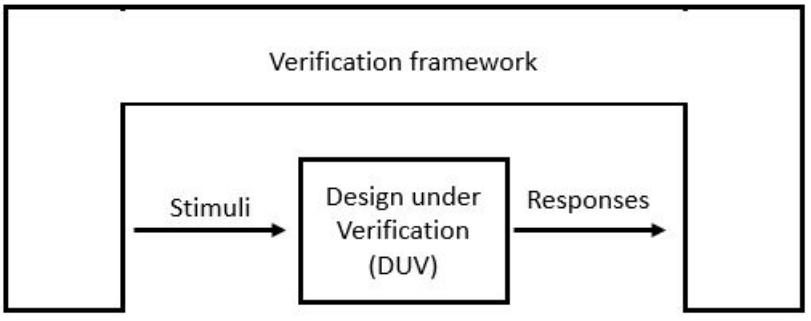

Figure 1.1: General verification process functionality

There are different verification types, each of them employs different techniques and implies its different benefits and drawbacks. In the following sections, a deeper approach of the most outstanding verification types used worldwide is presented.

In the first place, the direct verification is based in the application of different stimuli sets to the DUV, these stimuli sets must consider every possible case otherwise the given design could only

get partially verified which means the correctness of its behavior will be restricted to particular conditions. The principal advantage of this verification type is that it requires relatively little mental effort and short time lapses to generate each test. On the contrary, the excessive amount of time required to reach the verification completion is its main disadvantage since this computational time increases as high as the design complexion extends [3].

In the second place, the formal verification consists in the search of a mathematical expression set equivalent to a given design behavior. The principal benefit is that it only requires one test for the verification completion. However, the development of this test demands a great deal of mental effort and much more time than a direct verification test [7].

Finally, the functional verification involves random constraint stimulation which allows a performance enhancement in the simulation due to 3 main advantages. First, the coverage of different stimuli is done randomly which causes this process to be faster and more robust than in direct verification. Second, the amount of time and mental effort to generate a test are much less than in formal verification. Finally, the computational time to reach verification completion is dramatically less than in direct verification for high-level and medium-level designs. The only disadvantage refers to the fact that only for the generation of the first test the functional verification requires quite more time than the direct verification.

1.2 Problem description and formulation

In the digital design projects in this country, the direct verification has been used in the great majority of the time and in a reduced minority the formal verification instead. However, those verification approaches are certainly restrictive in the complexity level they can handle. This idea is clarified in this section through a comparison between the verification approaches previously described.

Generally, the low-level designs and some medium-level designs are characterized by a reduced number of input/output ports and a very predictive behavior. Consequently, there are only few or not so many different stimuli cases and high probability of finding equivalent mathematical expressions for their behavior. Those aspects contribute to the feasibility of using direct verification or formal verification. On the other hand, high-level designs and many medium-level designs normally implement many different input/output ports and a complex non-predictive behaviors. Hence, the number of possible stimuli cases increases drastically, the computational time required for the verification completion using direct verification becomes excessively high and there is enormous complexity on finding an equivalent mathematical

expression for the design behavior. This situation evidences the need for the implementation of a verification type in order to develop projects at a industry scale and overcome the limitations of the direct and formal verification [3]. This verification approach needed must be capable of guaranteeing time efficiency and high level of reliability in the verification process of medium-level and high-level designs, otherwise the projects scope will be restricted due to the lack of an adequate verification tool in this country.

1.3 Verification evolution and contextualization

The hardware design languages (HDL) have greatly contributed to the development of digital design since their creation. In the late 90s Verilog became the worlds most used HDL for simulation and synthesis. However, this HDL had simple constructions for verification which caused trouble to handle the verification process since the designs would constantly increment their complexity and size. This circumstances promoted the development of hardware verification languages (HVL) such as Openvera and e [3].

Shortly after, the developers on this area realized that the time utilized in both processes design and verification were close to each other which was denominated as the verification gap. The existence of 2 different languages for design and for verification and the rapidly development of the integrated circuits triggered the productivity crisis in this area [8]. In favor of finding the remedy for this crisis, Acellera, company of EDA, and external volunteers developed an HVL named **SystemVerilog** based on Openvera [3]. SystemVerilog could handle both design and verification, with consistent syntax and semantics, and the incorporation of object oriented programming (OOP) techniques. Hence, this HVL remarkably contributed to the overcoming of the verification gap and the productivity crisis [9]. Since its development, SystemVerilog has been used as a standard in the digital design projects although it has involved some challenges for the design and verification teams at first. A more specific study about these challenges and their impact is done in [8].

Currently, the universal verification methodology (UVM), which is based on SystemVerilog, is widely used in the industry for the verification process of most of the designs that are developed in the daily routine such as [1]. However, the employment of SystemVerilog remains mostly in the research area, an example of this is presented in [10]. This situation ensures the present research's viability in the use of SystemVerilog and the complexity level of the proposed design (AES encryption module) because its complexity level is similar to the one in [10]. At this point, it is important to highlight the fact that both works utilized functional verification and this is due

to the fact that in the countries where these works were developed the direct verification over the time has been relegated only for educational purposes.

In Peru, the GuE (Grupo de Microelectrónica) that operates in PUCP (Pontificia Universidad Catlica del Perú) can be considered as a reference in the digital design area. GuE is constituted by a group of people with research and training purposes in Microelectronics. Although GuE represents a high standard of development in this country, the direct verification has been employed in most of the projects GuE has developed since the beginnings. As a consequence, the scope of the projects has been restricted and the verification process has represented a very time-consuming activity.

The table 1.1 presents the verification method used in 7 thesis works developed within the last 9 years by GuE members in PUCP. Based on this table, it can be concluded that in the 100% of these thesis works the direct verification was employed to guarantee the validity of each design. However, not all the possible cases were tested during the simulation which reduces the reliability of this verification method. Consequently, under particular stimuli conditions any of those designs can experience failures. For this reason, the direct verification is not the best option to consider in the verification process of such designs though its employment is massively extended among the works in this country.

1.4 Importance and justification

The verification process is critical to the integrated circuit development since it is in charge of granting the validity and reliability level to any given design. Any failure in the verification method could end up in the erroneous validation of a defective design. Moreover, this design can even be massively produced and marketed before the error is identified which could cause serious consequences. Three of these consequences are listed below to illustrate the importance of the verification process.

- An international company can risk its reputation and lose its customers' confidence if one of its devices presents a failure in one of the internal circuits that was not detected during the verification process.
- Lives can be lost if any integrated circuit fails in a device that perform critical tasks in the medicine field.
- Valuable information related to banking or stock exchange can get seriously damaged in case one error in any circuit in the system shows up due to the fact that not all conditions were simulated for every circuit.

Table 1.1: Verification method used in 7 different thesis works developed in PUCP

Works	**Year**	**Verification Description**
[11]	2008	For the simulation of the entire system which includes the pre-processing modules, neural network and the post-processing modules, it was only employed one image stored in memory that corresponded to the digit 7. It is advisable for a robuster verification to test all 10 possible cases of images and evaluate delays and corner cases.
[12]	2008	The simulation of the complete system constituted by the direct digital synthesizer (DDS) and the audio module consisted of the assignment of a constant phase increment, consequently the wave generated in the first block was always the same. In order to get a higher level of reliability in the verification process, a random phase increment between the 2^{32} values should be selected in every different test.
[13]	2010	A function with 2 inputs and only 1 output was used to train the ANFIS system and get the precedent and consequent parameters. Besides, for the validation of this training, only 20 random values were employed. For a more solid verification a better option would be the employment of different functions in the training process and the use multiple random values and corner cases.
[14]	2011	For the validation of the FME quarter pixel architecture, it was used a MATLAB script to compare the obtained results. Moreover, 14 different sequences of distinct resolution (HDTV, OCIF or CIF) were utilized. A more complete verification process would have considered the stimulation of multiple sequences of distinct resolution with a random pixel distribution and the evaluation of delays.
[15]	2012	The verification process of each of the stages and FIR filter architecture modules consisted in the used of only one image with an OVGA resolution. A more consistent verification implies the employment of multiple random images with distinct resolutions and the consideration of delays between the images.
[16]	2013	The complete simulation of the system was based on the direct connection between the transmitter output and the receiver input, so the input bit-stream must be the same as the output bit-stream. The stimulus for the system was a random bit-stream of 4096 bits which was verified sample by sample. In order to get a robuster verification, it is advisable to employ multiple random bit-streams, considering corner cases and delays.
[17]	2014	The verification process of the conjugate gradient method implementation and the enhancement consisted of several tests with different sizes (powers of 2) between 8 and 512. Specifically, there were evaluated 10 tests of each size for a total of 70 tests. A complete verification would imply a stimulation with a greater number of tests for each size which must be selected randomly.

Another important point to highlight is that in Peru the digital design development is not based on the industry, it is restricted to the people and groups with research purposes that are reduced in number. Besides, in many of these groups there are not all the necessary tools (software, hardware or knowledge) to convert designs into functional integrated circuits. An adequate verification method is normally one of the missing tools for the validation of medium-level and high-level projects. Also, this method must be replicable and scalable so it can be used for the verification of any design even with industrial standards. Hence, this thesis work is aimed for the use of functional verification and its solid features that make it ideal to solve the problems and overcome the obstacles before mentioned in contrast to the traditionally used direct verification. Among these features, the most outstanding is the sufficiency in parameters of computational time and reliability in design at industrial scale such as [10] and [1]. The superiority of the functional verification over the direct verification is clearly shown in the Figure 1.2, where a comparison between them in terms of computational time is done.

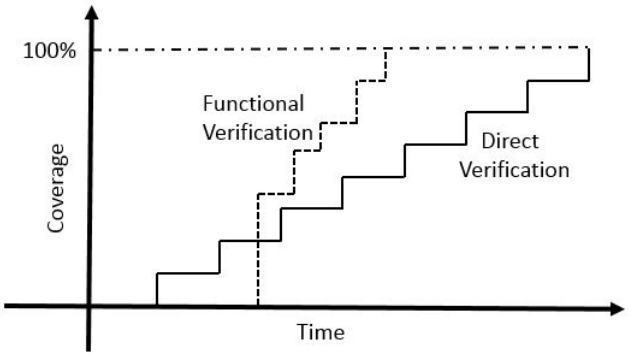

Figure 1.2: Coverage progress comparison between the functional verification and direct verification in terms of computational time (Adapted from [3])

Finally, it is required to set a precedent that demands a high verification level so the methodology employed can be reproduced in designs of all levels from low to high. For this reason, the design chosen to develop a pioneer functional verification environment is the AES encryption module. In general, the encryption modules require high standards of verification because of their complex structure and their common application on information security. Moreover, the AES is currently one of the most reliable encryption modules and it used worldwide [18].

1.5 Objectives

In general, the main objective of this thesis work is to generate a functional verification framework for an AES encryption module that uses the **Wishbone** bus interface which is in charge of performing the communication protocols between the DUV and the verification framework. Also, there are 4 specific objectives that this thesis work is going to cover and are listed below:

- Implement a functional verification environment described in SystemVerilog and based on all the specifications contained in the verification plan elaborated in the third chapter of this document.
- Define a functional verification methodology that can be reproduced in any given design of similar or lower complexity.
- Elaborate a verification plan template with technical quality standards similar to the actual industrial standards so it can be employed as a reference in any future project in PUCP.
- Write documentation about the study of SystemVerilog and develop simple and concise tutorials of its application so it can be easily utilized in any verification process.

Chapter 2

Approach to the functional verification and the AES encryption module

In this chapter, important concepts and basic notions about functional verification and the AES encryption module are explained in detail in order to provide an adequate comprehension of the solution presented in next chapters. Conveniently, this chapter has been divided in two different sections. the first section is related to the AES encryption module in general, that is to say its functionality, its algorithm and its requirements in order to identify the AES features and properties. The second section covers the functional verification methodology and the modular structure it employs as well as the behavior and usefulness of every module. Besides, in this second section the concepts of **Functional Coverage** and **Assertions** are described, these concepts constitute the base of the functional verification methodology.

2.1 Analysis of the AES encryption module

2.1.1 AES Algorithm

In 1997, the National institute of standards and technology (NIST) of the United States of America (USA) announced that a contest would take place to choose an algorithm as the new encryption standard which would be denominated as the AES (Algorithm encryption standard). For this reason, Joan Daemen y Vincent Rijmen developed an algorithm that they called Rijndael and presented it to the contest. Once the contest finished, this algorithm was chosen as the winner and ever after has been known in the industry as the AES [18].

The AES is a symmetric key algorithm in charge of the cryptographic processing of 128-bit data blocks with cipher keys that are multiple of 32 bits between 128 bits and 256 bits. For the

encryption process the inputs are the original text and the cipher key and the output is the ciphered text. On the contrary, for the decryption process the inputs are the ciphered text and the cipher key and the output is the original text. In each case the inputs and outputs are considered as one-dimensional arrays of 8 bits (1 byte) [18].

The AES is based on the application of a sequence of direct or inverse transformations for the encryption and decryption, respectively. However, the application of these transformations are not done over the data block, this block is first converted into a state block which is defined as a rectangular array with 4 rows and a variable number of columns (N_b) that is calculated by the expression 2.1. Consequently, the state block is a 4 X 4 rectangular array [18].

$$N_b = \frac{\text{Data block size (bits)}}{32 \text{ bits}} = \frac{128 \text{ bits}}{32 \text{ bits}} = 4 \tag{2.1}$$

Both the original text and the ciphered text are defined as one-dimensional and are represented as: $P = P_0P_1P_3...P_{4N_b-1}$ y $C = C_0C_1C_3...C_{4N_b-1}$, respectively; where the sub-index '0' denotes the first byte and '$4N_b - 1$', the last byte. The state block is denoted as: $a_{i,j},\ 0 \leqslant i < 4,\ 0 \leqslant j < N_b$, where *i* is related to the row number and *j*, to the column number. In Figure 2.1 it is described how the byte assignation is done in the pre-processing stage to generate the state block as well as the byte assignation for the ciphered data block during the post-processing; a similar byte distribution can be deduced for the decryption process [18].

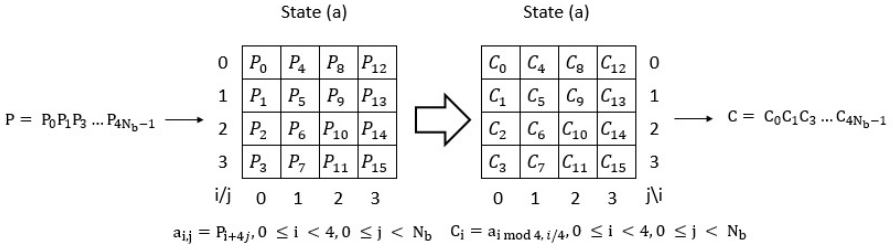

Figure 2.1: Byte assignation in the pre and post processing

Similarly, the input key is a one-dimensional array and is denoted as following: $Z = Z_0Z_1Z_3...Z_{4N_b-1}$. However, before the processing it is mapped onto a rectangular array with 4 rows and a number of columns denoted by N_k and equal to the key length divided by 32. This two-dimensional array of the cipher key is denoted as: $K_{i,j},\ 0 \leqslant i < 4,\ 0 \leqslant j < N_k$. The byte distribution of the input key onto the cipher key is defined in Figure 2.2.

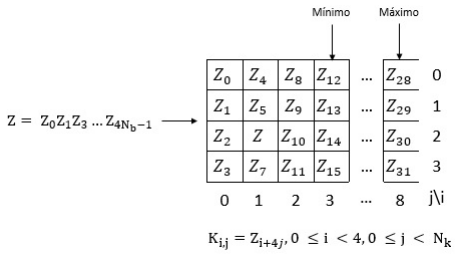

Figure 2.2: Byte assignation to the cipher key

Specifically, the encryption process is divided into 4 stages and those are: the expanded key generation (KeyExpansion function), the initial key addition (AddRoundKey function), the application of Round transformation N_r - 1 times, and the application of FinalRound transformation. The parameter N_r depends on the data block length and the input key length. Figure 2.3 shows high level flow diagram of the general encryption process using the AES algorithm [18].

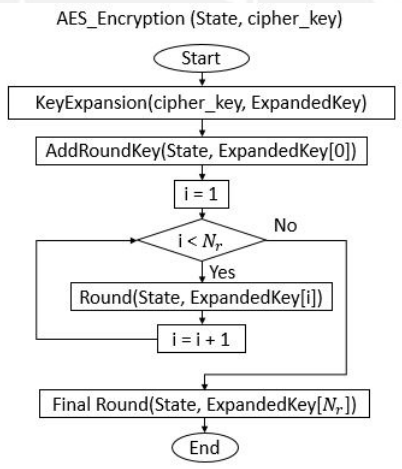

Figure 2.3: High level flow diagram of the AES encryption process

The sub-transformation KeyExpansion uses the key scheduler of the AES and generates the ExpandedKey array derived from the cipher key and constituted by N_r + 1 sub-keys that are used in each Round transformation [19]. The parameter N_r is defined according to the data length, so for this case it is recommended $N_r = 10$. The AddRoundKey sub-transformation takes as input the two-dimensional state array and a sub-key of the ExpandedKey array, then it modifies the state array by combining each of its bytes with the correspondent sub-key byte through a bit-to-bit XOR operation. This operation is possible due to the fact that both arrays are the same size. A comparison between the flow diagrams of the Round transformation (a) and the FinalRound transformation (b) is shown in Figure 2.4 in order to highlight that the only difference between them is the employment of the MixColumns function [18].

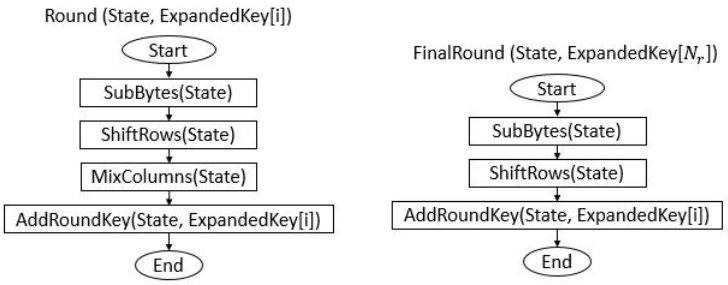

Figure 2.4: High level flow diagrams of the Round transformation (a) and the FinalRound transformation (b).

In the next paragraphs, a deeper approach of the functions mentioned before on the flow diagrams is done. In the first place, the SubBytes function consists of replacing each byte of the state array by the correspondent byte from a two-dimensional array of the same size denominated *S-box*. This array is generated from the application of two successive functions: a multiplicative inversion operation of finite field $GF(2^8) : g(a)$ which provides the non-linearity to the process and an invertible affined transformation: $f(a)$ which decreases the vulnerability to attacks based on mathematical properties. As a consequence, the calculus of the *S-box* values can be done as follows: $S - box[a] = f(g(a))$. Figure 2.6 describes the effect of the application of the SubBytes function to the State array [18].

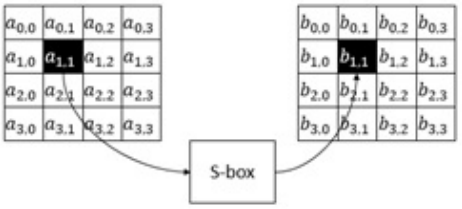

Figure 2.5: SubBytes application effect over the State array

In the second place, the ShiftRows function is based on a cyclic byte transposition in every row with different offsets. These offsets are established according to many probabilistic studies about the possible attacks that can occur. Figure 2.6 describes the effect of this function over the State array [18].

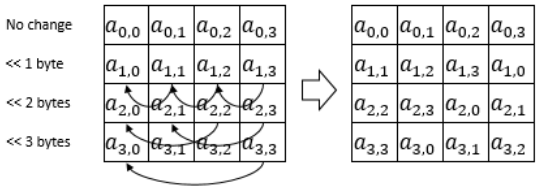

Figure 2.6: ShiftRows function effect over the State array

Finally, the MixColumns function consists of the application of an invertible linear transformation over the bytes in each column of the State array. Every State array column is considered a polynomial of $GF(2^8)$ and is multiplied in modulus by the non-reducible polynomial $x^4 + 1$ with the fixed polynomial: $C(x) = 3x^3 + x^2 + x + 2$. This operation is equivalent to the multiplication of every State array column by the fixed matrix denominated as *C*. In Figure 2.7, it is shown the MixColumns effect over the State array [18].

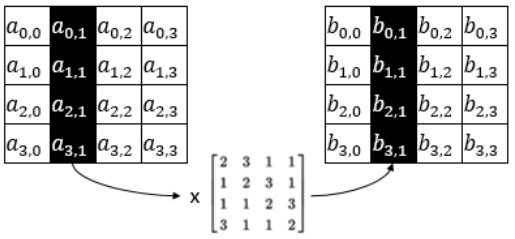

Figure 2.7: MixColumns effect over the state array

2.1.2 Design under verification

The AES encryption module that is used as the **DUV** (*Design Under Verification*) is specified in [2] and its source code was taken from the Opencores virtual repository, which is a foundation in favor of the easy access to open-source hardware designs. The communication between the DUV and the verification framework is performed by a Wishbone bus interface.

The Wishbone bus interface is a hardware module that implements a SoC (System on Chip) architecture and is defined as a portable interface with IP cores which means it does not require any permission or licence to use it [4]. In general, the Wishbone bus interface performs the communication according to the Master-Slave scheme and the data is sent in blocks through a handshake process with a maximum size of 64 bits [4]. In this thesis work it is employed a Wishbone controller adapted for only one master and one slave with data transmission in blocks of 32 bits, and it is based on the architecture obtained from [20]. A deeper approach of this hardware design is done on the first section of chapter 3.

2.2 Functional verification overview

2.2.1 Direct verification methodology

Traditionally, the direct verification has been employed in the verification process of any given design although its use has decreased over the time. This verification type implies the definition of a verification plan containing a list of tests so that each test can cover different corner cases, specific requirements or single properties and features. Also, the stimuli vectors need to be established for each test on this verification plan. Once the verification is complete, the simulation starts with the first test and the waveforms and output files are analyzed manually. When the simulation finishes, if the test was successful the next test should take place and the process start again. Finally, the design is fully verified only if all tests defined in the verification plan are successful [3].

This methodology is based on an incremental and progressive coverage so it only reaches the 100% of coverage when all possible cases are tested. For this reason, if the complexity of the design doubles, then the computational time of the simulation increases, greatly. As a consequence, this methodology can be successfully used with low-level designs which have reduced number of input/output ports and a predictive behavior such as: sequence detectors, adders and memory blocks. On the other hand, if this methodology is applied with medium-level or high-level designs which have multiple input/output ports and non-predictive behaviors, it can be easily deduced that its use is not feasible. For example, for the verification of a design with 10 1-bit inputs and 100 registers, using an average stimulation speed of 1000 stimuli vectors per second, the computational time required can be calculated as in (2.3).

$$Tcomputational = \frac{2^{10} \times 2^{100}\text{cases}}{1000\text{cases/s}} = 4,1 \times 10^{22}\text{years} \tag{2.2}$$

The previous calculus shows the limitation of the complexity level this methodology can handle. For designs of similar or higher complexity it is necessary to employ another methodology with time efficiency and high reliability.

2.2.2 Functional verification with random constraints

In the industry this type of methodology is greatly used for most of the designs from low to high level of complexity. In view of the fact that it allows fast error identification and considerably lower computational time to reach the full coverage in comparison with the direct verification [3]. The functional verification has 2 important metrics to measure the effectiveness of a verification framework and those metrics are: **Code Coverage** and **Functional Coverage**. The first one, code

coverage, is related to the percentage of the total lines that have been executed until a given moment and is important because during the simulation not all lines are executed due to conditional statements, which allow the execution of certain lines according to the logical value of an expression [5]. The second one, functional coverage, measure the proportion between the cases that have been exercised over the design until a given moment and the total cases. This is important because despite the fact that there are numerous cases, they can be grouped in reduced numbers according to the effect they produced in the design, thereby it is crucial to measure how many different cases are left to be exercised [3].

The functional verification methodology can be summarized on the next stages: random stimuli generation, driving direction of the stimuli to the DUV, output data monitoring, correct output data prediction according to a golden model (Reference model which implements the ideal behavior of the DUV) , comparison between the predicted and the actual results, progress evaluation according to the metric before described, and the alert generation in case of malfunctioning in the verification framework or its interface with the DUV which is normally done through **Assertions**. Each of these stages presents some specific details related to its implementation and usefulness, those details are listed below.

- Initially, the stimuli cases are restricted to the valid cases for the design and are group according to their effect on the design behavior. These groups of cases occur based on a probabilistic distribution that is set before the simulation starts and can be altered in order to evaluate specific cases or reach the maximum percentage of code coverage and functional coverage [21].

- Driving the stimuli vector to the DUV and Monitoring its outputs are processes that should be synchronized with the timing control of the DUV ports [3].

- The *Assertions* are monitoring tools to identify error occurrence during the simulation, it should be highlighted that an effective verification framework must employ assertions in favor of a constant performance evaluation of the DUV and the verification environment working together. Time efficiency and productivity can be affected if assertions are not utilized [22].

- For the verification process it is compulsory to own an ideal model of the behavior that is implemented by the DUV so both results can be compared at every stimulation during the simulation time [9]. This ideal model or golden model is not supposed to be a hardware implementation previously validated, but usually a software development implementing

desired behaviors [3].

2.2.3 Standard structure of the verification framework

The functional verification methodology demands a modular structure for the verification framework in which each module performs an specific task that could be expendable or indispensable according the DUV. In fact, in most of the low-level designs and some particular cases of the medium-level designs, not all the modules are required since some of them are considered optional. Besides, the modular structure is not definitive, it can employ different schemes and is highly adaptable to the particular features of a given design. This is motivated on the fact that a particular scheme can be the most effective for a design type, but it can present deficiencies if it is applied to other design type. This section will describe the standard modular structure used in the industry for the implementation of a verification framework. This standard structure is normally divided into 3 modules: **Test**, **Environment** and **Functional Coverage**. However, the environment module is sub-divided into 7 different sub-modules and those are: Generator, Agent, Scoreboard, Checker, Driver, Assertions and Monitor, the interconnection of these sub-modules is presented in Figure 2.8 [3].

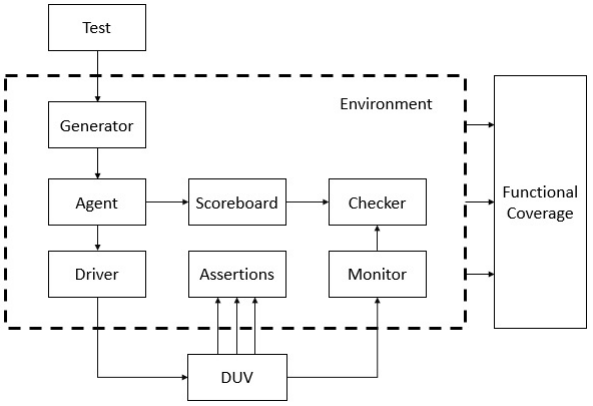

Figure 2.8: Standard modular structure of a functional verification environment (Adapted from [3])

The stimulation driven to the DUV is based on transactions, this is to say that the stimulation exercises one behavioral condition or a particular case at a time, and all data and configurations of an stimuli vector are specified in each transaction. Hence, Transactions are pieces of information that flow through most of the modules and sub-modules in the framework structure. Since all these modules and sub-modules have different features and perform different tasks, a general description with the most important aspects of each of them is presented below.

- Test: it has the highest hierarchy level and manages the the other 2 modules, also in it it is defined the general configuration of the complete environment.
- Generator: it generates randomly the transactions based on the random constraints and sends them to the agent sub-module.
- Agent: it receives the transactions from the generator sub-module, divides the transactions into single commands and sends them to the Driver and Scoreboard sub-modules.
- Scoreboard: it receives the commands from the Agent and predicts the appropriate results based on these commands and then sends the results to the checker sub-module.
- Checker: it receives the commands from the Monitor sub-module and the results from the Scoreboard sub-module and then compares both results in order to evaluate the success of the transaction.
- Driver: it has a direct connection with the DUV and receives the commands from the Agent sub-module, also it drives the appropriate signals to the DUV in the correct moment according to the commands previously received that refer to specific low-level tasks such as: Read, Write, reset, etc.
- Assertions: it monitors constantly the communication between the DUV and the verification framework, the assignment of valid values to the input ports of the DUV and the adequate reception of the DUV output values [22].
- Monitor: it has direct connection with the DUV and receives the output values that come from the DUV, grouping them into commands that are later sent to the Checker sub-module.
- Functional Coverage: it measures the progress of the transactions and stimuli sequences in favor of the completion of all requirements specified in the verification plan. At the end of the simulation, a report of the general performance is generated. [5].

2.2.4 Working methodology

The implementation of the functional verification framework which is the main objective of this thesis work has been generated following the next methodology in order to the set the adequate sequence of steps to successfully accomplish this objective.

- RTL analysis: inspection of the AES encryption module to determine the features and properties of the design, identify the requirements and comprehend its logic behavior over the time.

- Elaboration of the verification plan: setting the most important properties to verify, define all critical conditions and corner cases, determine the specific points where to use the assertions and the type of transaction that is required as well as the minimum level of functional coverage necessary.
- Implementation of the functional verification environment: create each of the modules and sub-modules of the modular structure, describing them in SystemVerilog and building all necessary internal interconnections.
- Verification process simulation: interconnect the DUV and the functional verification framework so they can be simulated together.
- Outcome evaluation: Compare the level of functional required and the actual level obtained, examine the results of all assertions during the simulation time and determine if the verification accomplished all the objectives described in the verification plan.
- Design Validation: Generate the final report containing the results of the verification and the level of reliability of the general process.

Chapter 3

Elaboration and implementation of the functional verification framework

3.1 Verification plan

3.1.1 Signal layer

The DUV requires a 128-bit key and 128-bit input data so the encryption/decryption process can be successfully done. For further comprehension a complete list of the DUV input/output ports is presented in Table 3.1. In addition, a detailed inspection of the DUV code and architecture has shown important specifications that are listed below.

- The S-box sub-module is implemented by circuitry, discarding the option of using a memory block containing defaults values.
- The design internally implements 3 Finite State Machines (FSM) which are distributed among the SubBytes, Keysched and MixColumn sub-modules.
- The DUV architecture does not implement the encryption/decryption process according to a pipeline methodology, where some tasks are performed in parallel. On the contrary, the process of a previous data block must finish before the next data block can be processed.

The Wishbone bus interface allows the communication between the DUV and the verification framework and as a consequence it has 2 different signal groups. The first group is used for the interaction with the DUV input/output ports considering the appropriate synchronization and timing. The second group is employed to interact with the verification framework according to the wishbone handshake protocol. Moreover, all signals in this second group are listed Table 3.2.

Table 3.1: Input/Output ports list of the DUV [2]

Signal	Direction	Length (bits)	Description
clk	Input	1	System clock
reset	Input	1	System reset
load_i	Input	1	Data load indicator
decrypt_i	Input	1	Encryption/Decryption indicator
data_i	Input	128	Input data (Original text or ciphered text)
key_i	Input	128	Encryption key
ready_o	Output	1	Operation completion indicator
data_o	Output	128	Output data (Ciphered text or original text)

Table 3.2: Wishbone controller input/output ports [4]

Signal	Direction	Length (bits)	Description
clk	Input	1	System clock
reset	Input	1	System reset
wb_stb_i	Input	1	Slave/Master indicator
wb_dat_i	Input	32	Input data
wb_adr_i	Input	32	Input address
wb_we_i	Input	1	Writing enable
wb_cyc_i	Input	1	Indicator of bus cycle validation
wb_sel_i	Input	4	Not utilized
wb_dat_o	Output	32	Output data
wb_ack_o	Output	1	Acknowledge

It is important to mention that this hardware module is used in this verification framework as glue logic which means it can be considered as ideal with no need to verify its behavior and is only used as an interface so the DUV and the verification framework can work together.

The handshake protocol implemented by the Wishbone bus interface is described in detail in Table 3.3. Besides, it is important to highlight that the lower layer on the verification framework should implement an adapter to connect the driver and monitor with the DUV. This adapter must perform accurately the handshake protocol in order to avoid any synchronization or interconnection problem. A diagram of the adapter implementation is presented in Figure 3.1.

Table 3.3: High-level handshake protocol of the Wishbone bus interface [4]

Order	Involved signals	Description
1	clk/reset	Bus synchronization
2	wb_stb_i/wb_we_i/wb_cyc_i	Bus write enabling
3	wb_dat_i/wb_adr_i/wb_ack_o	Control register transmission
4	wb_dat_i/wb_adr_i/wb_ack_o	Transmission of the input data in 4 32-bits blocks
5	wb_dat_i/wb_adr_i/wb_ack_o	Transmission of the cipher key in 4 32-bits blocks
6	wb_stb_i/wb_we_i/wb_cyc_i	Bus read enabling
7	wb_dat_o/wb_adr_i/wb_ack_o	Wait for the process completion
8	wb_dat_o/wb_adr_i/wb_ack_o	Reception of the cipher key in 4 32-bits blocks

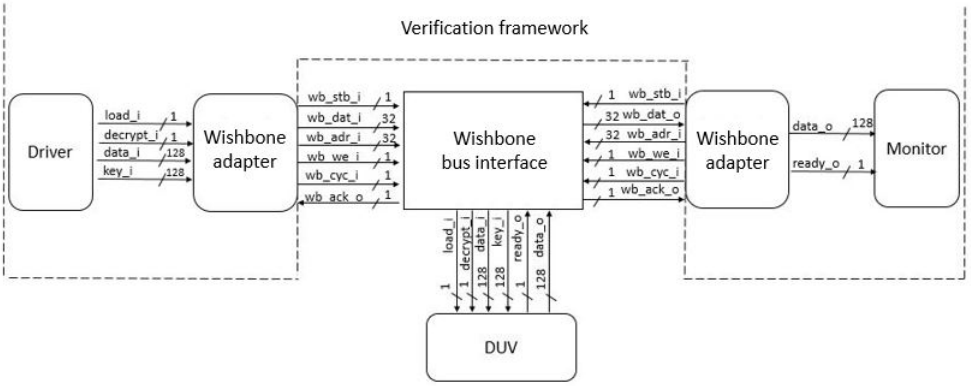

Figure 3.1: Diagram of the interconnection between the DUV and the functional verification environment

3.1.2 Stimuli sequences

In general, the stimuli sequences refer to the different test types that can be applied to the DUV according to its features and specifications. For the AES encryption module, 2 different stimuli sequences have been defined and are described in Table 3.4.

Table 3.4: Stimuli sequences for the AES encryption module

Sequence code	Description
Variant_data	The input data and cipher key are generated randomly in order to verify the fact that the encryption and decryption process should work for any given pair of input data and cipher key.
Invariant_data	Only the cipher key is generated randomly while the value of the input data remains the same during a pre-established time period. The objective of this test type is the verification of the effectiveness in the encryption and decryption process for multiple keys using the same data.

3.1.3 Validation mechanism

The Environment module of the verification framework is composed by a Scoreboard sub-module which is in charge of predicting the correct result based on the current stimuli vector. Hence, it requires a golden model, in other words, a model that implements the expected behavior of the DUV with no errors. At the industry, this model is normally a piece of software described in a programming language such as C/C++ or a system description language such as SystemC. For this reason, it should be developed an adapter between the HVL (in this case SystemVerilog) and the golden model, so it can be correctly instanced in the verification framework.

The golden model employed in this thesis work is an adaptation in C/C++ of the SystemC model contained in the AES encryption module file downloaded from the OpenCores online repository [2]. Furthermore, the interconnection between this C/C++ model with the verification

framework in SystemVerilog has been possible due to the employment of the Direct Programming interface (DPI) from SystemVerilog. The DPI allows a SystemVerilog model to call any function or construction from a foreign language such C/C++ and some others. In particular, the DPI-C plays the roll of an adapter between C/C++ and SystemVerilog [3].

A significant aspect to point out is that the golden model employed presents some limitations since it is not a piece of hardware, on the contrary it is software based. An analysis of the golden model has encountered 3 main limitations in contrast to the DUV and are presented below.

- Constant arrays with default values are used to perform operations in the SubBytes and MixColumn sub-modules.
- There are two functions per sub-transformation, one for the encryption process and another for the decryption process in contrast to the DUV behavior with only one sub-module per sub-transformation.
- The decryption process is based on a flow diagram different from the one defined by the AES standard in [2].

3.1.4 Coverage metrics

Functional coverage is a metric of verification effectiveness and it involves a report of all the requirements (covered by cover-points/cover-groups and assertions) defined for a given DUV which specify a property or a particular feature of the design such as signal timing, response time, signal toggle, etc. In addition, this requirements are assigned with a requirement code label so it can be easily identified in the functional verification report. Furthermore, the type of verification used to cover the requirement and the priority are defined for every requirement along with the requirement code. Table 3.5 presents the complete list of requirements for the AES encryption module.

Another useful metric is the Code coverage and it measures how much of the code in the DUV has been exercised by the functional verification framework. Most of the software tools for verification and design present the code coverage report at the end of simulation. Generally, this metric is divided into 5 different types and each of these types has particular methodologies to test the performance in the simulation. In Table 3.6, those types are listed along with their description and the minimum percentage ratio expected to be reached during the simulation. It is important to notice that at the industry the standard percentage ratio for all types of code coverage is between 95% and 100%, so in order to define a functional verification framework capable of handling designs at industry scale, this thesis work takes this standard as a requirement.

Table 3.5: Requirement specifications of the DUV

Requirement code	**Description**	**Validation type**	**Priority**
load_i_val	After the posedge of load_i, it must not occure a new posedge of load_i, before the DUV process completeness	Property	1
data_i_val	If load_i is High, input data must only be 1s or 0s, No Xs or Zs	Property	1
key_i_val	If load_i is High, key_i must only be 1s or 0s, No Xs or Zs	Property	1
data_o_val	If ready_o is High, output data must only be 1s or 0s, No Xs or Zs	Property	1
data_i_stb	data_i must remain stable after the posedge of load_i, until the negedge of load_i	Property	1
key_i_stb	data_i must remain stable after the posedge of load_i, until the negedge of load_i	Property	1
data_o_stb	data_o must remain stable after the negedge of ready_o, until the posedge of load_i	Property	1
decrypt_i_stb	wb_decrypt_o must remain stable until the posedge of ready_i	Property	1
dti_adr_seq	data_adr must change its value according to a pre-established sequence during the simulation	Property	1
dto_adr_seq	data_o_adr must change its value according to a pre-established sequence during the simulation	Property	1
kyi_adr_seq	key_adr must change its value according to a pre-established sequence during the simulation	Property	1
ready_o_tm	ready_o must only be High during 1 clock cycle	Property	1
load_i_tm	load_o must be in High just during 1 clock cycle	Property	1
time_out_tm	The time limit between a posedge of load_i and a posedge of ready_o is 500 clock cycle	Property	2
fsm_stuck_cst	None of the FSM used in the AES module should get stuck for any stimuli vector	Property	3
data_i_cvr	data_i must cover at least 100 value in every sub-range of the 128 sub-ranges	Cover-point	2
key_i_cvr	key_i must cover at least 100 value in every sub-range of the 128 sub-ranges	Cover-point	1
case_stim_cvr	Both stimuli sequences must be covered at least 40% of the time, the least probable to occur	Cover-point	2
decrypt_cvr	decrypt_i should cover both cases, at least 40% the least probable of them	Cover-point	2

Table 3.6: Code coverage metrics

Type	Description	Expected ratio (%)
Expression	How many statements were covered	100
Block	How many blocks have been covered	95
Toggle	How many times signals and ports are toggled during simulation	95
FSM	Indicates whether the FSMs in the design reach all possible states or not	95

3.2 Implementation review

In chapter 2, it was discussed the standard structure of a functional verification framework and a general description of each module and sub-module contained in this structure was presented. In this section, the specification of these modules and sub-modules for the verification of the AES encryption module is described in detail.

3.2.1 Test module

The Test module represents the top most level in hierarchy of the verification framework and it contains some important definitions such as the number of tests and the instance of the Environment module as well as its methods. Moreover, according to the the DUV, multiple tests are defined in order to exercise all DUV features and each of these tests is based on a different stimuli sequence. However, the verification framework for the AES module defines only one type of test due to the fact that a macro overview of this module shows that its functionality is based on a response in its output ports after an stimulation on its input ports with any variant which requires no need to develop multiple tests. Nonetheless, this verification framework takes into account 2 different stimuli sequences which were specified in the verification plan, but these stimuli sequences are handled directly in the randomization process of the transactions. Despite of the fact that there is only one test type for this verification framework, it is exercised as many times as the times set on the Test module, so for each time it is required a new Transaction with a different stimuli vector randomly generated.

As it was established in the verification plan, the total range for the data_i and key_i has been divided into 128 sub-ranges and the minimum number of occurrences per range to validate the DUV is 100. There are two important aspects that determined this decision in the current verification framework.

First, a functional verification framework is not based on exercising all possible cases, it must be focused on analyzing if all intended functionality of the design have been verified and this involves defined transitions, variable ranges, multiple states evaluation, corner cases, signal timing,

etc. For the AES, all particular requirements has been defined in the previous section and the corner cases are limited since according to [23] there are no weak or semi-weak key values, so all possible key values have the same probability to fail. Besides, most transformations that the AES module performs are linear and the only non-linear transformation takes place in the SubBytes sub-module and it is a bricklayer permutation which means that neither the data value nor the key value has impact over the general behavior [2]. In conclusion, exercising only a segment of the whole range of possible values can ensure the validation of the DUV for any value within the possible range as long as all requirements are fulfilled.

Second, in the industry functional coverage and code coverage are two important metrics to determine the efficiency and consistency of a given verification framework, but there are other important aspects to consider when the verification performance is analyzed. These aspects refer to the memory occupied and the elapsed time during simulation, since normally the unit-level verification frameworks which verify single circuits are not simulated individually. On the contrary, they are used in system-level verification frameworks that verify complex systems, so the minimum consumption of memory and time are always important requirements. For this reason, a good verification framework requires trading off the reliability level against the resource consumption. This means the number of tests must be high enough to reach an acceptable reliability level and low enough to consume the least possible resources.

This verification framework implements two cover-points for the data_i and the key_i signals which have 128 bits each, and the established number of sub-ranges is 128 with at least 100 occurrences per each, so the minimum number of test to validate this DUV is 12800. However, this is an ideal number since in the randomization process many values can be repeated and the invariant_data stimuli sequence increments the probability of repetition. Hence, this verification framework implements a control over the values through a sub-range signal which is randomized cyclically and contributes to reach the occurrences required faster, with a number of tests that is very close to the double of the minimum. An analysis of how the reliability level and the elapsed time during simulation change with higher or lower number of sub-ranges is presented in Chapter 4.

Finally, it has to be mentioned that the Test module represents the highest level in hierarchy of the verification framework, but this module as well as the DUV and any glue logic employed are instanced on the Top module which is the main module and is normally used to instance all interfaces and resources, and is in control of the clock generation.

3.2.2 Environment module

The Environment module is considered as the core of the verification framework since in this module all Transactions are generated, driven to the DUV and evaluated. Each transaction contains all information of a given stimuli vector which involves stimuli sequence, random constraints, input signal values, auxiliary signals or variables, expected results and the actual results from the DUV so it can be determined whether the stimulation was successful or not. In addition, transactions are defined based on the DUV, so all specific parameters and considerations for the transaction used in the current verification framework of the AES encryption module are shown in Table 3.7.

Table 3.7: Transaction parameters [3]

Parameter	**Description**
Stimuli Sequence	Defines if the current transaction exercises Variant_data or Invariant_data, this second sequence is possible because in every transaction the data_i value is temporally stored in the prev_data signal.
Random Constraints	Both key_i and data_i are 128-bit long so they can represent numbers from 0 to $2^{128} - 1$, so this enormous range has been conveniently divided into 128 sub-ranges. For this reason, these signals take their value according to the value of a Sub-range signal that indicates which of the 128 sub_ranges is exercised.
Input signal values	There are 3 signals: data_i, key_i and decrypt_i which hold the actual values that are exercised in the DUV during the current transaction.
Auxiliary signals	There is an id variable which identifies the number of transaction and a static id counter which is in charge of granting the correct value to the id variable. Also, the sub_range signal previously discussed is defined.
Expected results	This parameter refers to the output of the Golden model located in the Scoreboard sub-module and the signal holding this value is: data_o_scb
Actual DUV results	This parameter refers to the output of the DUV collected in the Monitor sub-module and the signal holding this value is: data_o_duv.

All Environment sub-modules are directly or indirectly interconnected to each other, so the data can easily flow through the entire module. There are different techniques to reach this objective and the most common are described below.

- Callback: involves pre_transmission and post_transmission methods that are called in the transmitter class before and after a particular the call to a particular task or function. Those methods take the data required to be sent as input, and then call a task from this receiver class to save this data into a Queue declared on it.

- Direct handler: Based on calling a task directly from a class, sending the data required as inputs. However, this technique is only valid if the receiver class was declared inside the transmitter class which normally does not occur.

- Mailbox:it is a direct interface that can be used between 2 independent classes to transmit and receive one or more signals of a pre-established data type.

The verification framework for the AES module implements Callbacks and Mailboxes, since it presents a modular structure where all sub-modules are treated as independent classes with no option to use the Direct handlers. All Callbacks and Mailboxes used in this verification framework are presented in Table 3.8.

Table 3.8: Interconnection interfaces used in the verification framework

Interface	**Type**	**Sub-modules/modules interconnected**
mbG2A	Mailbox	Generator and Agent
mbA2D	Mailbox	Agent and Driver
mbM2C	Mailbox	Monitor and Checker
Dcbk	Callback	Driver and Monitor
Acbk	Callback	Agent and Scoreboard
Ccbk	Callback	Checker and Coverage

Another important aspect to take into account is the synchronization and timing in the verification framework. During the simulation, most tasks are executed in parallel, but some of them must stop execution until an external condition is given. In consequence, a synchronization protocol is required to manage all concurrent and sequential tasks. In the industry, there are some techniques that are normally employed such as semaphores and events which are explained below.

- Semaphores: implement mutually exclusive access to a resource with multiple requestors from inside the verification framework. However, most tasks in the present framework do not manipulate the same resources since a modular structure with multiple different tasks is implemented.

- Events: Are handlers to synchronization objects that can be passed through routines with no need to declare them globally. Basically, their functionality is based on flag objects that can be triggered to unblock a part of the code that required a given condition.

This verification framework employs only events to manage the synchronization among the multiple tasks execution during simulation. Those events are specified in Table 3.9 as well as the sub-modules involve.

Table 3.9: Interconnection event interfaces used in the verification framework

Event	Sub-modules/modules involve
DoneG2A	Generator and Agent
DoneA2D	Agent and Driver
DoneD2M	Driver and Monitor
DoneM2C	Monitor and Checker
DoneA2S	Agent and Scoreboard

In general this thesis work follows the standard modular structure for the functional verification. However, for further comprehension, the modular structure defined for the AES verification framework and all specifications are shown in Figure 3.2.

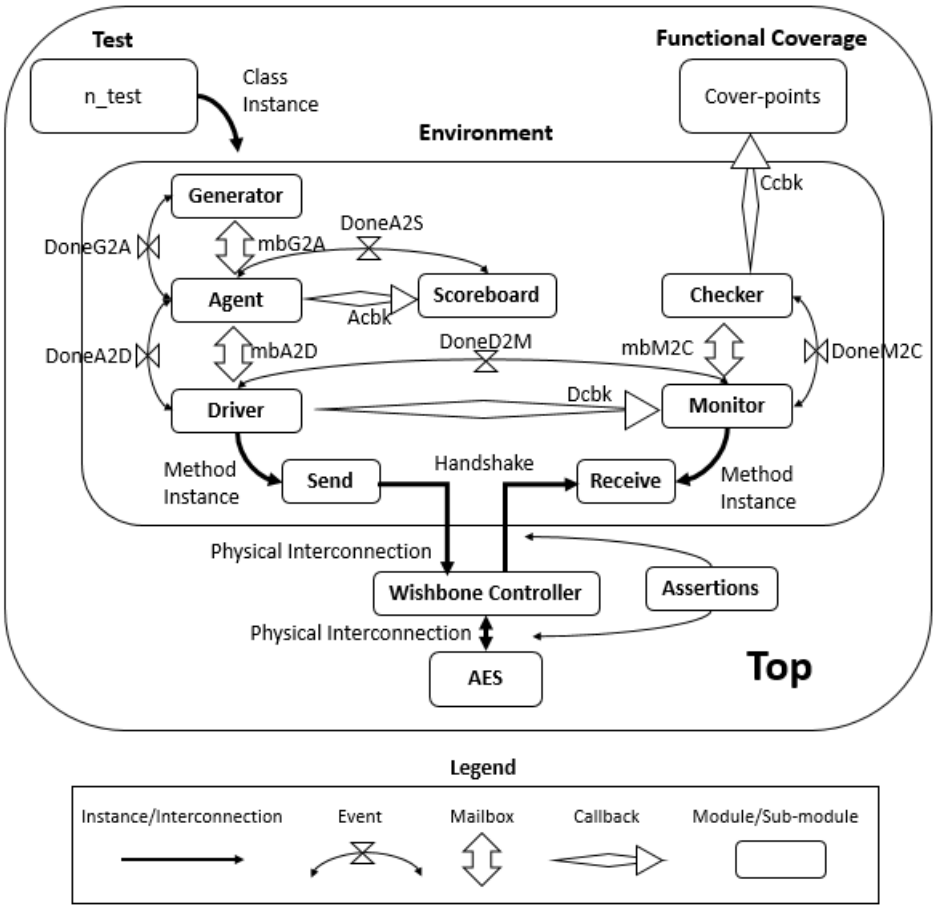

Figure 3.2: Modular structure of the functional verification framework for the AES encryption module - Own design

The Generator sub-module is in charge of the randomization process of the Transactions. This process evaluates two aspects before assigning a value to each signal in the stimuli vector. First, the value for each signal is restricted to the range defined in the random constraints, only if no constraints are defined for a given signal, any possible value can be assigned to it. Second, each

signal is randomized according to its data type, it means that the types rand and randc are evaluated differently. When a rand type is evaluated, in each randomization any possible value considering the random constraints can be obtained. On the contrary, when a randc type is evaluated, in each randomization not only the random constraints restrict the possible value, but also the values that have been obtained before, since all possible values should be obtained before any of them can be selected again, which is denominated as a cyclic randomization. In Table 3.10, each signal of the stimuli vector with its respectively data type and a brief argument for the decision taken are presented.

Table 3.10: Randomization types in the stimuli vector signals

Signal	**Data type**	**Description**
data_i	rand	Any possible value is valid
key_i	rand	Any possible value is valid
decrypt_i	rand	There are only 2 different values, so a cyclic randomization will be much monotonous
stim_seq	rand	There are only 2 different values, so a cyclic randomization will be much monotonous
sub_range	randc	Cyclic randomization takes much less tests to reach the goal of 100 occurences per sub-range

Once the randomization process is finished, the transaction generated is sent to the Agent sub-module using the mailbox mbG2A and then generator's execution is blocked until the event DoneA2G is triggered by the Agent sub-module.

The Agent sub-module is in charge of sending the Transaction handler to the Driver sub-module and the Scoreboard sub-module, for this purpose the mailbox mbA2D and the Callback Acbk are used, respectively. As soon as the handler is sent, the execution gets blocked until the event DoneA2D as long as the event DoneA2S are triggered by the Driver and Scoreboard sub-modules, respectively. When this last condition occurs, the event DoneA2G is triggered so the Generator can continue its execution.

The Driver sub-module is located at the lowest layer of the verification framework and it is directly interconnected with the DUV. For this reason, this sub-module should implement an adapter to perform the handshake protocol appropriate for the DUV. First, the Driver sub-module receive the Transaction handler from the Agent sub-module through the Mailbox mbA2D. Second, the data in the transaction is divided into single commands which are sent to the DUV using the wishbone adapter. Finally, this sub-module execution gets blocked until the event DoneM2D is triggered by the Monitor sub-module, then the event DoneA2D is triggered so the Agent can continue its execution.

The Scoreboard sub-module predicts the expected results for a given stimuli vector, so first it

receives the Transaction handler trough the callback Acbk, then the input data is sent to the Golden Model using the DPI-C from SystemVerilog. Soon after, the results are obtained and stored in the Transaction parameter of expected results. Finally, the event DoneA2S is triggered so the Agent can continue its execution.

The Monitor sub-module as well as the Driver sub-module is located at the lowest layer of the verification framework and requires an adapter. First, the handler to the Transaction is received through the callback Dcbk and the transaction parameter of actual results is filled with the output collected from the DUV. Second, the Transaction handler is sent to the Checker sub-module using the mailbox mbM2C. Finally, the execution is blocked until the event DoneM2C is triggered by the Checker and once this happens, the event DoneD2M is triggered so the Driver can continue its execution.

The Checker sub-module compares the result from the DUV and the Golden Model and determines whether the Transaction was successful or not. It is important to highlight that this verification framework not only checks for the final result in the DUV to be correct, but also checks every partial result from the DUV which refers to the results at every Round transformation. As a consequence, if any error is encountered in the partial results, then the Transaction would be considered as failed. In addition, in order to simplify the error identification when testing a design, for every Transaction a report is printed in console containing all stimulation information and the evaluation result. An example of this report is presented in Figure 3.3. Besides, as soon as the evaluation finishes, the event DoneM2C is triggered so the Monitor can continue its execution. Finally, the Transaction handler is sent to the Coverage module using the callback Ccbk.

```
-------------------------------------------------------------------------------
at         11253951500: -------------- TRANSACTION N*       20499 ----------------------
-------------------------------------------------------------------------------
---------------------------- Has been successfully driven -----------------------------
-------------------------------------------------------------------------------
 ------------------- DATA_IN is: b3fe1f2342b3e09d5f8251544f54cf17 ----------------------
 ------------------- KEY_IN is: b35f68750b8d15968c8b9b8ab6e78ceb  ----------------------
-------------------------------------------------------------------------------
--------------------------------- PROCESS: ENCRYPTION ---------------------------------
-------------------------------------------------------------------------------
 ----------------- DATA_OUT_SCB is: 0be8ba0a1e404432c5f7d9d1bffc2f06 -------------------
 ----------------- DATA_OUT_DUV is: 0be8ba0a1e404432c5f7d9d1bffc2f06 -------------------
-------------------------------------------------------------------------------
---------------------- All partial results were correct! ------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------------- In TRANSACTION N*       20499 : NO Errors Encountered ----------------
-------------------------------------------------------------------------------


--------------------------- COVERAGE SUCCESSFULLY SAMPLED -----------------------------
-------------------------------------------------------------------------------
```

Figure 3.3: Transaction report

About Figure 3.3, the report presented in this Figure gives detail of the stimulation and the results at some point during the simulation. The first line specifies the elapsed time until the stimulation takes place and the number of transaction it is. The second line states whether or not the stimulation has been successfully driven to the DUV. The third an Fourth lines show what value of ”data_in” and ”key_in” are being evaluated during the present stimulation. the fifth line determines the type of process that is being exercised (Encryption or Decryption). The sixth and seventh lines shows the results obtained from the golden model and the DUV, respectively. In the eighth line a message is printed indicating if the partial results were correct, if only one of them is wrong then an error message is printed instead. In line tenth the number of error encountered in the present stimulation is printed. Finally, in the line eleventh it is indicated whether or not the stimulation has been successfully sampled.

3.2.3 Functional Coverage

The Coverage module receives the Transaction handler trough the callback Ccbk and then it samples the data on it according to what was previously defined in the verification plan. As it was discussed before, the coverage is based on cover-points which should be associated with the signals of interest. In this verification framework, five signals of interest were defined and each of them requires different coverage options. In general, a coverage option defines a parameter of a cover-point or a cover-group, so it gives the chance to the verification engineer to set different coverage configurations for each of the cover-points/cover-groups defined in the verification plan. All different coverage options employed in the AES verification framework are described in detail in Table 3.10.

Table 3.11: Coverage options employed [5]

Coverage option	**Description**
Auto_bin_max	Defines how many bins (specific ranges or values of interest) are generated automatically when no bins are declared
Weight	Defines the priority to reach the appropriate number of occurrences for a given cover-point/cover-group
at_least	Defines the minimum number of occurrences required to consider a cover-point/cover-group as fully covered

For further comprehension, all cover-points defined in the verification plan as well as the coverage options values established for each of them are listed in Table 3.12.

Another important aspect of the functional coverage is the inlclussion of assertions which were defined, as well as the cover-points in the verification plan. The assertions were not

Table 3.12: Cover points specifications

Coverage option	**id_cover**	**decrypt_cover**	**data_i_cover**	**key_i_cover**	**case_stim**
Auto_bin_max	-	-	128	128	-
Weight	2	2	1	1	2
at_least	-	-	100	100	12500

considered as an independent module. All of them were declared inside the top module since the all assertions evaluate signals in the bus interface declared in the top module, and another module instance was not required. In general, the assertions are aimed to verify an specific requirement of the verification plan and to build them, reusable sequences were defined which increases the reusability level of the code. Moreover, both the requirement and the assertion related to this requirement hold the same name to make their relation obvious. Nevertheless, the assertions that end with the index ”aux” were defined to reduce complexity in some assertions. Besides, the assertions are classified according to its purpose and the way to identify them is noticing the last value on each name which can vary among different indexes. A further explanation of each index is detailed in Table 3.13.

Table 3.13: Coverage options employed

Assertion index	**Description**
val	verifies the validity of the data at some point in the simulation time
stb	verifies the stability of the data when driving stimuli or monitoring results
seq	verifies that all transitions between sequences are correct
tm	verifies if the appropriate amount of clk cycles has passed before any further event occurs
cst	verifies the consistency of state machines

Chapter 4

Simulation and result evaluation

4.1 Design handling

The DUV asserts the signal ready_o whenever the result is ready and stored in the output port data_o, this signal remains asserted for one clock period. However, at the end of every Round transformation the corresponding partial result is stored in the output port data_o, but the ready_o signal is not asserted when this event occurs. Hence, there is no way to identify when the partial results are ready so the verification framework can read their value. In order to solve this problem, a modification was made in the DUV which allows the assertion of the signal ready_o in every partial result. In Figure 4.1 the signal waveform of all ports in the DUV before the modification is shown and in Figure 4.2 the same waveform is presented after the modification implementation.

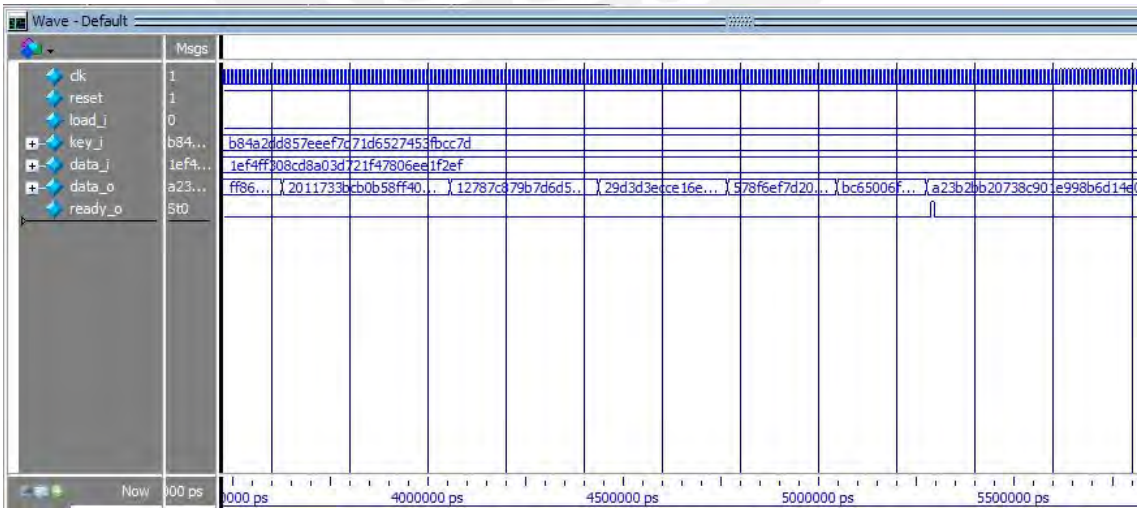

Figure 4.1: DUV behavior before the modification

Figure 4.2: DUV behavior after the modification

According to previous chapters, the implementation of two Wishbone adapters was required in order to build a intercommunication with the DUV. These adapters are located in the lowest layer of the verification framework and the result of their performance evaluation is presented in Figure 4.3. In this figure, it can be appreciated that the handshake protocol is implemented successfully (the protocol uses all signals that begin with the index wb) and the DUV responds to the stimuli, calculating the partial results (This occurs at every positive edge of the signal "ready_i") and obtaining the final output data (This occurs at the 11th positive edge of the signal "ready_i" which is highlighted in yellow). The final result is stored on the output port of the DUV and collected by the wishbone controller.

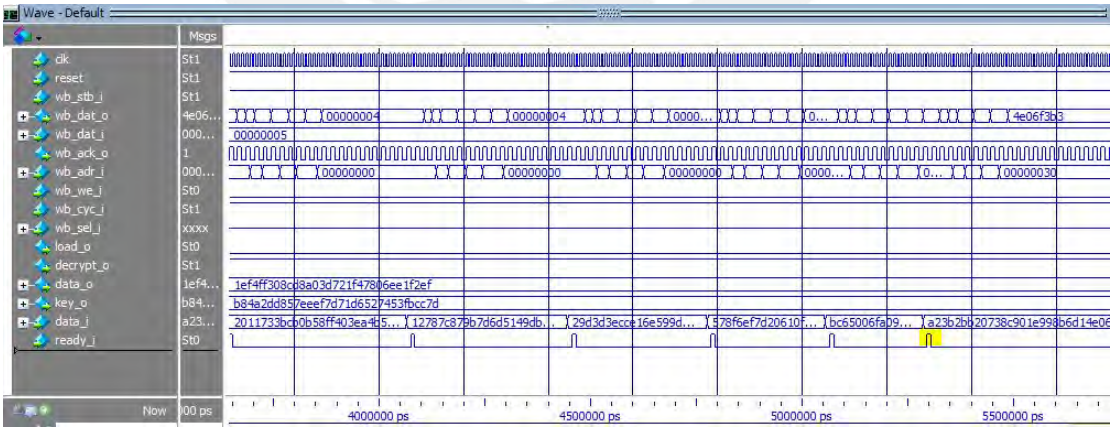

Figure 4.3: DUV and Verification framework intercommunication

So far, the wishbone bus implementation as the intercommunication interface between the DUV and the verification framework has been accomplished and no problems were detected.

4.2 Golden model handling

The Golden model employed in this work is a C/C++ adaptation of the a SystemC-model obtained from the Opencores online repository. The adaptation consisted not only of building equivalent instructions and statements from the SystemC model to the C/C++ model, but also on altering the algorithm in order to get the partial results for every Round transformation. During the simulation of the golden model, the results are printed in the console. In Figure 4.4, the simulation of the altered golden model is presented and it can be seen that the new algorithm allows to collect every partial or final result.

```
VSIM 51> run
# In Golden Model:
#
# For data_i    = 1ef4ff308cd8a03d721f47806ee1f2ef
# For key_i     = b84a2dd857eeef7d71d6527453fbcc7d
# For decrypt_i = 1
# The result N*           1 is 5aa5417524288ae7cdb9feb594ce5215
# The result N*           2 is 9369389b4a88dc4d97b86bc0a252d6f0
# The result N*           3 is a8c8e5c1a1d613396859593aecfb8532
# The result N*           4 is cb43cbaac871c1be603ec708cb8a2fec
# The result N*           5 is ff8687e6c102d8587e2c6fb4c8460a9d
# The result N*           6 is 2011733bcb0b58ff403ea4b58987be4a
# The result N*           7 is 12787c879b7d6d5149db98f572acefa8
# The result N*           8 is 29d3d3ecce16e599d01f199484bc5316
# The result N*           9 is 578f6ef7d20610f896e4ec94a8ac560a
# The result N*          10 is bc65006fa09090ab0f4ffcd15cfdf8af
# The result N*          11 is a23b2bb20738c901e998b6d14e06f3b3

VSIM 52>
```

Figure 4.4: Golden model simulation

4.3 Coverage results

As it was established in chapter 3, the number of tests depends only on the times required for the completion of all design requirements and it was determined that the number was close to the double of the minimum (12800 times). This analysis was done based on 128 different bins (number of sub-ranges of all possible values) for the key_i and data_i input ports. Consequently, during simulation it was determined that the appropriate number is 20500 since all requirements and coverage statistics were accomplished at this number of times. Something important to mention at this point is that in the industry normally the simulation are not restricted to a number of tests, most of the time million of tests are exercised for better reliability. However, in this work the number of tests is restricted since the goal is to reach a high level of reliability considering the least resources (memory and time) consumption. Hence, the final report considering this number is shown in Figure 4.5.

```
at:        11254500500: ------------ VERIFICATION FINISHED ----------------------------
-------------------------------------------------------------------------------------
---------------------------------- VERIFICATION REVIEW -------------------------------
------------------------- Transactions delivered =       20500 ----------------------
------------------------- Errors Found =             0 ----------------------------------
-------------------------------------------------------------------------------------
---------------------------------- TEST PASSED --------------------------------------
-------------------------------------------------------------------------------------
Simulation complete via implicit call to $finish(1) at time 112545005 NS + 3
```

Figure 4.5: Verification report summary

Based on this simulation the analysis of code coverage and functional coverage are presented in the next section.

4.3.1 Code coverage

The code coverage results of the complete DUV specified by code coverage type can be appreciated in Figure 4.6 and the average grade of all code coverage type for each sub-module is presented in Figure 4.7.

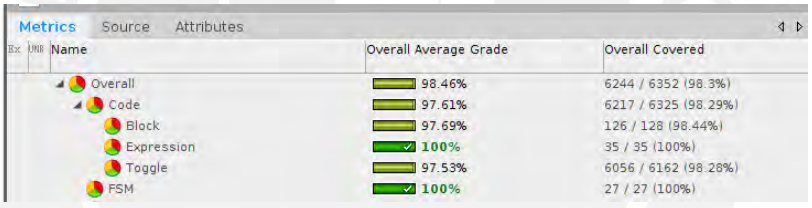

Figure 4.6: Code coverage by type

Figure 4.7: Average grade of code coverage in each module

Based on these results, all requirements of code coverage were accomplished according to the verification plan (greater than 95%), reaching 100% in all code coverage types is hard to accomplished since it does not only depend of the verification framework, but also it depends of

the design and the codification techniques employed. Moreover, with more tests the results can improve slightly.

4.3.2 Functional verification

The functional coverage consists of cover-points and assertions, both results are summarized in Figure 4.8 and further performance detail is given in Figure 4.9 and 4.10 for the cover-points and assertions, respectively.

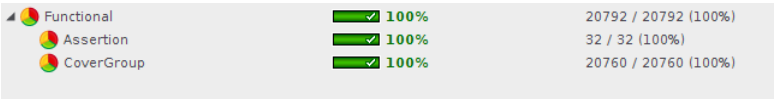

Figure 4.8: Functional verification summary report

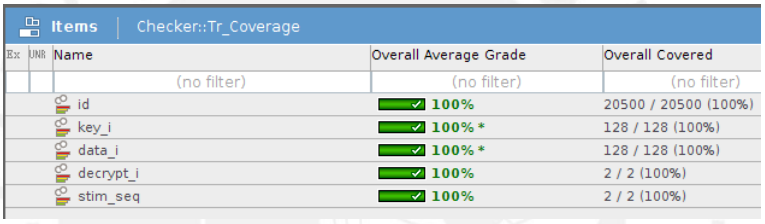

Figure 4.9: Cover-points coverage performance

These cover-points elements were defined in order to fulfill the requirements established in the verification plan, more specifically the requirements whose codes end with the index ”cover”, so the cover-point element ”id” responds for the id_cover requirement and so on. Based on Figure 4.9 all bins were covered for each cover-point at 100%, which means that all the related requirements were successfully covered.

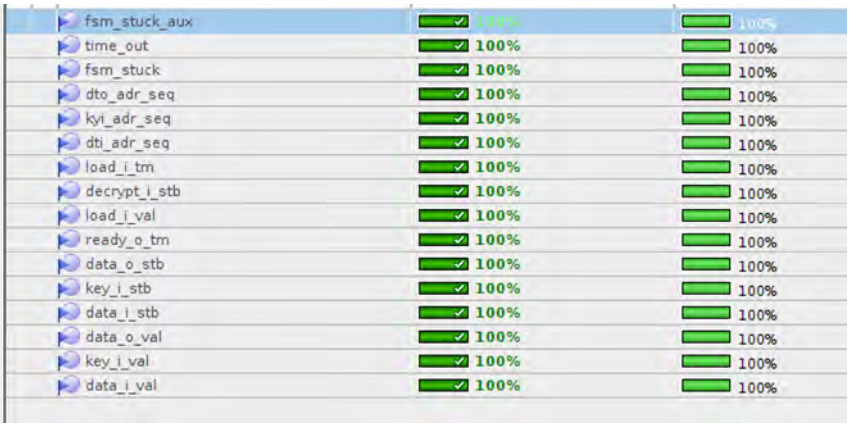

Figure 4.10: Assertions coverage performance

The assertions are evaluated at least once during each transaction, so if the number of times an assertion was evaluated is less than this number, it has failed despite of the fact that it might be successful in all evaluations. For this reason, a more specific report is presented in Figure 4.11.

```
ncsim> assertion -summary
  Disabled Finish Failed   Assertion Name
         0  20500      0   top.data_i_stb
         0  20500      0   top.data_i_val
         0  20501      0   top.data_o_stb
         0 225500      0   top.data_o_val
         0  20500      0   top.decrypt_i_stb
         0  20500      0   top.dti_adr_seq
         0 225500      0   top.dto_adr_seq
         0  20501      0   top.fsm_stuck
         0  20501      0   top.fsm_stuck_aux
         0  20500      0   top.key_i_stb
         0  20500      0   top.key_i_val
         0  20500      0   top.kyi_adr_seq
         0  20500      0   top.load_i_tm
         0  20500      0   top.load_i_val
         0 225500      0   top.ready_o_tm
         0  20501      0   top.time_out
  Total Assertions = 16,  Failing Assertions = 0,  Unchecked Assertions = 0
  Assertion summary at time 112545005 NS + 3
```

Figure 4.11: Assertions occurence report

Based on the report of Figure 4.11, all transactions were evaluated at least once per Transaction (number of Transactions = 20500) and were successful each time so the requirements were successfully accomplished.

4.3.3 Reliability vs Resources Consumption

As it was discussed in Chapter 3, the reliability level a verification framework can reach depends on the resources available, so the higher the reliability level is intended, the most resources are employed. In order to illustrate this point an analysis is presented where different number of bins and tests are evaluated. It is important to mention that this analysis is focused only on the computational time during simulation since with memory the analysis is similar and the evaluation does not have to be very rigorous as it is just an illustration.

Before further discussion of this analysis, it must be pointed out that all simulations in this research have been restricted to the memory and clock frequency of the computational system in which they were evaluated. In table 4.1 the specifications of this computational system are listed.

Table 4.1: Computational system specifications

Parameter	**Value**
Processor	Intel(R) Core(TM) i3-4700MQ
Clock frequency	1.2 GHZ
RAM	2 Gigabytes

In table 4.2 the data required for this analysis is presented and it has been collected from

different simulations of the verification framework design in Chapter 3. It should be noticed that only 4 simulations are included in Table 4.2 and this is due to the RAM size, which is normally the principal limitation in the number of bins.

Table 4.2: Simulation comparison

Number of simulation	Number of bins	Number of tests	Computational time (s)	Times of the fastest simulation
1	128	20500	112.55	5.12
2	96	15000	82.35	3.75
3	64	8400	46.11	2.09
4	32	4000	21.96	1

About Table 4.2, an important aspect to understand is that the most number of bins are considered, the most reliable is the verification. Although an ideal number would consider all possible values, this is pointless since many stimuli vectors have the same effect over the DUV, so the possibilities are divided in sub-ranges and the most sub-ranges quantity, the closest to the total. On the contrary, in simulation it is important to get reduced computational time and memory consumption so the less they are consumed, the most efficient the verification framework is. Based on these ideas it can be concluded that the simulation 1 was the most reliable and simulation 4 was the most time-efficient. Besides, although the number of bins increases by 32 in every simulation, the computational time does not increase linearly as the cost of time gets much bigger. The next simulation with 160 bins would take at least 7 times the first simulation and with further simulations this difference will increase drastically. For this reason in the industry a trade-off is made between resources and reliability. In this verification framework the number of bins selected was 128 since the computational time that 160 bins would require does not make it worth the little increment in reliability.

4.3.4 Comparison highlights

In [24], a verification framework is developed to verify an AES encryption module which implements the same architecture as the DUV of the present research. This section is oriented to compare the metrics and reliability of these works. It is important to mention that these works implements different verification methodologies, consequently the performance of theses verification framework differ from each other. There are three important aspects to highlight.

First, the performance held by the verification process using the methodology proposed in this work presented higher metrics of code coverage in comparison with the one obtained from [24] which is summarized in Table 4.3.

Table 4.3: Code Coverage metrics comparison

Code coverage type	**Result in this work (%)**	**Result in [24] (%)**
Expression	100	99.57
FSM	100	100
Block	97.69	94.51
Toggle	97.53	84.85

Second, in contrast of the work in [24] which presented no results of functional coverage, the present work reach 100% in all cover-points and assertions as it was presented in Figures 4.8, 4.9 and 4.10.

Finally, the structured of the proposed methodology facilitates the specification of a more robust analysis in opposition to the work in [24] as it can be seen in Table 4.4.

Table 4.4: Code Coverage metrics comparison

Functional verification feature	**Quantity in this work**	**Quantity in [24]**
Assertions	15	7
Cover-points	5	5
Total bins	20760	316
Goal average per bin	540	1

Conclusions

- The verification plan for the AES encryption module was developed according to the technical quality standards used in the industry to ensure a high reliability level. Based on the performance metrics of code coverage and functional coverage obtained during simulation, all requirements defined in the verification plan were fulfilled and there were no failures neither in the implementation nor in the simulation. In view of these outcomes it can be stated that the verification framework was completely successful. (Appendix 1)

- The methodology proposed verified an AES encryption module completely and presented no problems to evaluate all design requirements from the verification plan. The modular structure facilitated the employment of a glue logic hardware as interconnection interface which can be easily replaced with similar hardware due to its reusability.Further validation of this methodology is done in the appendix 4 where a different design (a floating Multiplier-Accumulator) is entirely verified using the proposed methodology. For these reasons, it can be stated that this methodology can be reproduced on the verification process of different designs independently of the architecture they implement.

- The verification plan defined in this work is based on a standard structure and technical specifications used in the verification industry (Advisor from ARM). It includes the definition of stimuli sequences, transactions, validation mechanism and coverage metrics which allow the accomplishment of high levels of quality and reliability. The appendix 2 contains the template of the verification plan which can be used with multiple types of designs with similar or lower complexity of the AES.

- The appendix 3 contains the documentation of the study of SystemVerilog which is based on all the information collected during the elaboration of this research, containing explanations and useful examples to facilitate the use of SystemVerilog focused in verification. Besides, a verification of a floating point Multiplier-Accumulator (MAC) is done in appendix 4 with more detail to illustrate the sturdiness of the present verification methodology.

Recommendations

- In order to exercise the reusability of the modular structure methodology, a different bus interface should be employed to interconnect the DUV and the verification framework such as the APB bus.
- The AES encryption module is normally used as a module stage of a System on Chip (SoC) design, a further work should develop a system level verification framework using the present unit level verification framework as a sub-module to evaluate the scalability of this work.
- With the purpose of testing the effectiveness of the verification methodology proposed, many other design must be verified with different features and behaviors. For instance, it can be used in the development of future projects of the research group in Microelectronics PUCP (GuE PUCP).
- The evaluation of error injection to the DUV is an aspect not developed in the methodology proposed and it would be useful for a more robust verification. However, it should be previously established in the verification plan the desired behavior when the error occurs.
- An improvement of the present work is aimed to develop a Sequencer module to select randomly multiple types of Transactions instead of handling the stimuli sequences internally, which will allow the verification methodology to verify designs of a higher level of complexity.
- The upgrade of the present methodology should be implemented in UVM which is the most recent standard of verification in the industry and it involves high level of reusability with many pre-defined blocks and innovative tools for verification. Basically, this upgrade requires to find the UVM equivalent blocks and tools used in this methodology, this adaptation is not quite complex since UVM is based in SystemVerilog.

Bibliography

[1] E. Gamst and E. Mitacc, "Verification of a large heterogeneous many-core computer," Master's thesis, Norwegian University of Science and Technology (NTNU), 2016.

[2] J. Castillo, "128 AES verilog module." `http://www.opencores.org/projects.cgi/systemcaes/`. Accessed: 2017-06-15.

[3] C. Spear, *SystemVerilog for verification: a guide to learning the testbench language features*. Springer Science & Business Media, 2008.

[4] R. Herveille *et al.*, "Wishbone system-on-chip (SOC) interconnection architecture for portable IP cores," *OpenCores Organization*, 2002.

[5] A. Piziali, *Functional verification coverage measurement and analysis*. Springer Science & Business Media, 2007.

[6] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[7] R. Drechsler *et al.*, *Advanced formal verification*, vol. 122. Springer, 2004.

[8] D. Rich, "The unique challenges of debugging design and verification code jointly in systemverilog," in *Specification & Design Languages (FDL), 2013 Forum on*, pp. 1–7, IEEE, 2013.

[9] J. Bergeron, *Writing testbenches using SystemVerilog*. Springer Science & Business Media, 2007.

[10] R. Sethulekshmi, S. Jazir, R. A. Rahiman, R. Karthik, M. Abdulla, *et al.*, "Verification of a risc processor ip core using systemverilog," in *Wireless Communications, Signal Processing and Networking (WiSPNET), International Conference on*, pp. 1490–1493, IEEE, 2016.

[11] M. A. Monge Osorio, *Diseño de una arquitectura para una red neuronal artificial perceptron multicapa sobre una FPGA aplicada al reconocimiento de caracteres*. Pontificia Universidad Católica del Perú (PUCP), 2011. Tesis para optar la licentura de Ingeniría Electrónica.

[12] T. Seclen and J. Lucio, *Diseño de un modulador FM basado en la tecnología software-defined radio en FPGA*. Pontificia Universidad Católica del Perú (PUCP), 2014. Tesis para optar la licentura de Ingeniría Electrónica.

[13] H. J. Block Saldaña, *Diseño de una arquitectura para un sistema neurodifuso ANFIS sobre un FPGA aplicado a la generación de funciones*. Pontificia Universidad Católica del Perú (PUCP), 2011. Tesis para optar la licentura de Ingeniría Electrónica.

[14] E. C. Villegas Castillo, *Diseño de una arquitectura para la interpolación de quarter-pixel para estimación de movimiento según el formato H. 264/AVC empleado en el estándar SBTVD de televisión digital terrestre*. Pontificia Universidad Católica del Perú (PUCP), 2011. Tesis para optar la licentura de Ingeniría Electrónica.

[15] C. E. Cano Salazar, *Diseño de una arquitectura de un filtro digital de sobre muestreo de imágenes, en factor 2, de acuerdo al formato H. 264/SVC sobre FPGA*. Pontificia Universidad Católica del Perú (PUCP), 2012. Tesis para optar la licentura de Ingeniría Electrónica.

[16] M. Meza and E. Máximo, *Diseño de un sistema de transmisión/recepción basado en OFDM para comunicaciones PLC de banda ancha*. Pontificia Universidad Católica del Perú (PUCP), 2014. Tesis para optar la licentura de Ingeniría Electrónica.

[17] S. A. Sosa Cordova, *Implementación del método gradiente conjugado en un FPGA arquitectura Spartan 6*. Pontificia Universidad Católica del Perú (PUCP), 2014. Tesis para optar la licentura de Ingeniría Electrónica.

[18] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[19] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.

[20] M. Sharma and D. Kumar, "Wishbone bus architecture-a survey and comparison," *arXiv preprint arXiv:1205.1860*, 2012.

[21] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based verification*. Springer Science & Business Media, 2006.

[22] A. B. Mehta, *SystemVerilog assertions and functional coverage: guide to language, methodology and applications*. Springer, 2016.

[23] N. F. Pub, "197: Advanced encryption standard (AES)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[24] H. Ruud, "Verification of an aes rtl model with and advance object-oriented testbench in systemverilog," Master's thesis, Norwegian University of Science and Technology (NTNU), 2007.