REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

UNIVERSAL VERIFICATION METHODOLOGY CASE STUDY ON FIELD PROGRAMMABLE GATE ARRAY FOR HARDWARE CERTIFICATION IN AVIATION

İbrahim AYAZ

MASTER OF SCIENCE THESIS
Department of Avionics Engineering
Avionics Engineering Program

Supervisor
Assoc. Prof. Dr. Ufuk SAKARYA

Co-supervisor
Dr. İbrahim HÖKELEK

February, 2022

REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

UNIVERSAL VERIFICATION METHODOLOGY CASE STUDY ON FIELD PROGRAMMABLE GATE ARRAY FOR HARDWARE CERTIFICATION IN AVIATION

A thesis submitted by İbrahim AYAZ in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 14.02.2022 in Department of Avionics Engineering, Avionics Engineering Program.

Assoc. Prof. Dr. Ufuk SAKARYA
Yıldız Technical University
Supervisor

Dr. İbrahim HÖKELEK
TÜBİTAK BİLGEM
Co-supervisor

Approved By the Examining Committee

Assoc. Prof. Dr. Ufuk SAKARYA, Supervisor
Yıldız Technical University ____________________

Dr. Onur AKBATI, Member
Yıldız Technical University ____________________

Prof. Dr. Sıddıka Berna ÖRS YALÇIN, Member
Istanbul Technical University ____________________

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of Universal Verification Methodology Case Study on Field Programmable Gate Array for Hardware Certification in Aviation supervised by my supervisor, Assoc. Prof. Dr. Ufuk SAKARYA. In the case of a discovery of false statement, I am to acknowledge any legal consequence.

İbrahim AYAZ

Signature

Dedicated to my father Dr. Mehmet AYAZ,
my mother Dr. Selma AYAZ,
my sister Kevser AYAZ,
and my cat, Sutlac

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my thesis advisor Assoc. Prof. Dr. Ufuk SAKARYA and my co-advisor Dr. İbrahim HÖKELEK who guided me with their valuable knowledge and experience throughout my master thesis work. This thesis wouldn't have been done without their valuable contributions.

I would like to thank my project manager Hakan YILMAZ and my mentor Yıldız ULUDAĞ from TÜBİTAK BİLGEM for their contributions that they have provided to me throughout my thesis.

I would like to thank TÜBİTAK BİLGEM TUTEL for opening their laboratory environment for the case study of this thesis.

I would like to express my deepest thanks to my father Dr. Mehmet AYAZ who made me love the aviation since I was a child and who has worked in the aviation industry for many years, to my mother Dr. Selma AYAZ who is a researcher and has a PhD in the environmental engineering that gave her love and support throughout my thesis, to my sister Kevser AYAZ who is always with me with her suggestions and love, and lastly to my cat Sutlac who relieved my stress in the darkest times.

İbrahim AYAZ

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMC	Acceptable Means of Compliance
ARINC	Aeronautical Radio Incorporated
ARP4754A	Guidelines for Development of Civil Aircraft and Systems
ARP4761	Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
ASIC	Application Specific Integrated Circuit
DAL	Design Assurance Level
DO-178	Software Considerations in Airborne Systems and Equipment Certification
DO-254	Design Assurance Guidance for Airborne Electronic Hardware
DUT	Design Under Test
EASA	European Union Aviation Safety Agency
ED-80	Design Assurance Guidance for Airborne Electronic Hardware
EDA	Electronic Design Automation
EUROCAE	European Organisation for Civil Aviation Equipment
FAA	Federal Aviation Administration
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
PHAC	Plan for Hardware Aspects of Certification
PLD	Programmable Logic Device
RTCA	Radio Technical Commission for Aeronautics
SC-167	Special Committee of the RTCA for the Software

SC-180	Special Committee of the RTCA for the Hardware
SDI	Source/Destination Indicator
SOI	Stage of Involvement
SSM	Sign/Status Matrix
UVM	Universal Verification Methodology
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WG-46	Special Committee of the EUROCAE for the Hardware

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Universal Verification Methodology Case Study on Field Programmable Gate Array for Hardware Certification in Aviation

İbrahim AYAZ

Department of Avionics Engineering
Master of Science Thesis

Supervisor: Assoc. Prof. Dr. Ufuk SAKARYA
Co-supervisor: Dr. İbrahim HÖKELEK

As the complexity of modern avionics system increases, the safety becomes the most critical aspect for both the military and civil aviation applications. Therefore, every hardware which will be developed for any aircraft must be designed with the compliance of DO-254 processes. These processes include activities and objectives that necessary to ensure the safety. According to DO-254, there should be some extra verification activities made to cover safety needs for DAL A and B custom micro coded component designs. Requirements are the key elements of the aviation. All the requirements must be covered to a design to count as completed. Therefore, verification of the custom micro coded components against requirements should be taken care comprehensively.

The most time-consuming process among Programmable Logic Device (PLD) design processes is the verification. Therefore, choosing the right verification method is a highly critical decision for making sure that the safety requirements are met while the project budget is not exceeded. Human made verification is less preferable method as humans can make mistakes. Therefore, the most used verification method today is the automated simulation. The industry has developed a common methodology that everyone can create different automated testbenches with the same way. In this way reusability of the verification environments are maxed. This methodology's name is the Universal Verification Methodology (UVM).

In this thesis, theoretical information of the DO-254 processes are explained. Universal Verification Methodology (UVM) is used to verify ARINC – 429 data bus as a digital subsystem. This verification activity is presented by complying DO-254 verification process as requirements are written and verified.

Keywords: DO-254, UVM, ARINC-429, aviation, certification

ÖZET

Havacılıkta Donanım Sertifikasyonu İçin Universal Verification Methodology Kullanımı İle Field Programmable Gate Array Üzerinde Vaka Çalışması

İbrahim AYAZ

Aviyonik Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Assoc. Prof. Dr. Ufuk SAKARYA
Eş-Danışman: Dr. İbrahim HÖKELEK

Günümüz aviyonik sistemlerinin gün geçtikçe karmaşıklaşması ile askeri ve sivil havacılık uygulamalarında emniyet konusu en önemli konu haline gelmiştir. Bu nedenle, herhangi bir hava aracı için geliştirilecek olan tüm donanımlar DO-254 süreçlerine uyumlu bir şekilde tasarlanmalıdır. DO-254 süreçleri anlattıkları aktiviteler ve görevler ile emniyeti sağlamaktadır. DO-254'e göre, özel mikro kodlu bileşen tasarımlarında emniyetin sağlanabilmesi için bazı ek doğrulama faaliyetleri yapılmalıdır. Gereksinimler havacılığın temel unsurlarıdır. Bir tasarımın tamamlanmış sayılabilmesi için bir tüm gereksinimleri karşılanmalıdır. Bu nedenle, özel mikro kodlu bileşenlerin gereksinimlere göre doğrulanmasına kapsamlı bir şekilde özen gösterilmelidir.

PLD tasarım süreçleri arasında en çok zaman alıcı olan süreç verifikasyon sürecidir. Zaman ve bütçe anlamında doğru verifikasyon metodunun seçilmesi çok kritiktir. İnsan dahili verifikasyon havacılıkta daha az tercih edilen bir yöntemdir çünkü insanlar hata yapabilir. Bu nedenle günümüzde en çok kullanılan doğrulama yöntemi otomatik simülasyondur. Endüstri, farklı tipteki otomatik testbench'lerin aynı yol ile oluşturulabilmesini sağlayan ortak bir metodoloji geliştirmiştir. Bu sayede doğrulama ortamlarının tekrar kullanılabilirliği maksimuma çıkarılmıştır. Bu metodolojinin ismi UVM'dir.

Bu tez çalışması kapsamında DO-254 süreçlerine ilişkin teorik bilgiler anlatılmaktadır.

ARINC – 429 veri yolunu dijital bir alt sistem olarak doğrulamak için Evrensel Doğrulama Metodolojisi (UVM) kullanıldı. Bu verifikasyon aktivitesi, DO-254'de aktarılan verifikasyon sürecine uygun gereksinimler yazılması ve bu gereksinimlerin doğrulanması ile gerçekleştirildi.

Anahtar Kelimeler: DO-254, UVM, ARINC-429, aviation, certification

1
INTRODUCTION

1.1 Literature Review

In the early days of digital design, verification was a time consuming task since it was done primitively by analyzing the waveforms. However, due to the increase in complexity of today's designs, verification must be performed in the form of automation. A new coding language is needed to make this possible, because Verilog and Very High Speed Integrated Circuit Hardware Description Language (VHDL) are not sufficient for the complex verification activities. Although the simulation environment simulates the hardware, the SystemVerilog language [1] has been invented by taking many advantageous features from generic software languages because the testbench code is compiled at the verification side.

Electronic design automation tool manufacturers developed their own verification environments using the SystemVerilog language in order to provide reusable, robust, and scalable verification. UVM has been created by combining these environments with the common work of electronic design automation tool manufacturers. UVM is open source so that everyone can use the same environment for hardware design verification. It is developed by Accellera, a nonprofit organization, and the UVM library can be accessed from the Accellera's website [2]. One of the important aspects of UVM is that it is an IEEE standard. With the last update in 2020, UVM's version number is IEEE 1800.2-2020 [3].

In general, application specific integrated circuit (ASIC) design path is more challenging than the field-programmable gate array (FPGA) design path. As ASICs are designed specifically for the application, they are more efficient and consume much less power than FPGAs. However, today, if the ASIC chip production is not in large scale (e.g., millions), it poses great difficulties for companies in terms of budget and time. If a design is related to aviation and aircraft, then FPGAs are more preferred in this field because of less number of production requirements.

According to the aviation safety regulation, any technology that has not been approved by the aviation authorities cannot be integrated into the aircraft. The documents created by aviation authorities provide certain process and standard management according to the type of the technology. Among these documents, the safety is considered as the most important aspect. The name of the document that is expected to be followed in electronic hardware design for the safety compliance is DO-254 [4].

1.2 Objective of the Thesis

Processes are explained at a very high level in the DO-254. That's why the aviation authorities are publishing additional documents to clarify the processes to the companies and personnels that aim to develop airborne electronic hardware. Even this additional documents may not be sufficient. This study aims to expound DO-254 processes at the focus of the verification process and to develop a case study which clarifies the verification activities to verify a digital design.

1.3 Hypothesis

In this thesis, the digital design and verification studies of ARINC-429 data bus with the compliance of DO-254 processes using Universal Verification Methodology (UVM) is presented. The coverage data is demonstrated with the use of UVM along with constraint random verification and the implementation of automated tests. The digital design which will be verified by UVM is selected as ARINC-429 since it is the frequently used data bus in the avionics applications. This thesis presents not only UVM but also the DO-254 processes and why it is important to perform the verification in the form of automation.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows. The background is presented in Section 2. The case study is described in Section 3 and finally, the results and discussions are provided in Section 4.

2
BACKGROUND

2.1 Design Assurance Guidance for Airborne Electronic Hardware / RTCA DO-254

DO-254 is a guidance document which stands for Design Assurance Guidance for Airborne Electronic Hardware. The purpose of the document is to ensure that the electronic design to be carried out on airborne platforms is safe within the framework of the established rules. The DO-254 document does not describe how an electronic design should be done. Instead, it describes how an electronic design can be done safely by applying standard processes and rules. It reveals that a safe design will emerge as a result of the audits carried out whether certain tasks are completed and properly observed. DO-254 is mostly applied for complex custom micro coded components(e.g., ASICs, FPGAs, and Programmable Logic Devices-PLDs) with Design Assurance Level (DAL) A, B, and C.

2.1.1 Design Assurance Level

There are many electronic equipment in the aircraft, where each one is dedicated to a specific function. While some hardware may be designed to perform safety critical tasks, other hardware may be designed to perform less critical tasks. An example of a safety critical system is the flight computer. An example of a low critical level system is the in-flight entertainment system. The scientists who prepared the DO-254 document, balanced the load that would occur while performing processes, according to their criticality levels. They sorted and named the criticality levels of the hardware as Design Assurance Level (DAL). They ranked DALs with A being the most critical and E being the least critical. The functions of a DAL A hardware are very critical, and in case of a failure, may result in loss of life. For this reason, DAL A is categorized as "Hazardous". The equipment that needs to be designed at DAL E has no effect on the flight safety, therefore is categorized as “No effect”. For DAL E hardware, it is unnecessary to implement the DO-254 processes, however for DAL A hardware, it is mandatory to apply the processes and the objectives from the main text of the DO-254 and the Appendix-B of DO-254.

2.1.2 History of the DO-254

DO-178 [5] is the guideline document for the software development in aviation which named as Software Considerations in Airborne Systems and Equipment Certification and was published long before than the DO-254 document. Equipment manufacturers, who wanted to overcome the difficulties of the processes required by the DO-178, preferred to create their equipment only by designing the hardware, without including the software. In this way, they would get serious relief in terms of time and budget, as they would not have to implement the DO-178 processes. However, as a result of this situation, hardware designs that are not complied with the design assurance had emerged. To prevent this undesirable situation, the DO-254 document was published by RTCA, by making use of the DO-178 document, in 2000 by a group of experts in the field, including scientists and industry people which named as SC-180. RTCA which stands for Radio Technical Committee for Aeronautics [6] is a non-profit organization in the aviation industry that pioneers new technology methods. The FAA [7] recognized that the document should be followed as a means of compliance in 2005, 5 years after the document's release. There is also a document created by EUROCAE [8] which named as ED-80 [9] which is a parallel document to DO-254. Participants, who wish, can also define and present their own processes to the certification authority rather than following the DO-254. However, this method is not recommended as it will be prohibitively challenging and expensive.

2.1.3 Aviation Authority Regulations

Applicants who are planning to design airborne electronic hardware for civil aviation based platforms should obey the Aviation Authority regulation documents which are called as "Acceptable Means of Compliance". These documents are written by both FAA and EASA [10] which are the Aviation Authorities from the United States of America and the Europe respectively. AMC contains the objectives, alongside the guidance and the best practices [11]. Completing the AMC objectives is the most important aspect where participant provides the proof to the Aviation Authority that the design is made in the scope of all the rules which should be followed. The participant, who completes all the DO-254 objectives, is assumed to complete the AMC objectives, therefore AMCs recognize the DO-254 document as the design guideline. The DO-254 document provides guidance to all types of hardware, however its main focus is the complex hardware which are typically PLDs.

2.1.4 Requirements

The most important task in the aviation industry is writing correct and complete requirements to provide safety. Electronic hardware is designed and audited through the requirements while verification is applied according to the requirements. There is no sense to apply perfect verification activities if the requirements are not correct. The hardware which is designed and verified with the wrong requirements is not appropriate to the Aviation Authorities and the customer's expectations. That's why requirements are the most important subjects for providing safety.

Requirements should tell the designer "what to do" rather than "how to do". There are certain rules and guidelines for writing requirements in the aviation industry [12]. Getting used to these rules and applying them correctly requires serious practice for both staff and the companies. The rules have been created with the experience of people and companies that have worked in this field for many years. Implementing these rules to the company's requirements standard document provides significant benefits and is highly recommended.

Requirements should be designed as traceable to the other levels with the usage of the required tools. Requirements should be organized and grouped by functions of the hardware. The person who writes the requirements should have the detailed knowledge of the system which contains the hardware and the sub-components which the hardware contains.

2.1.4.1 PLD Requirements

PLD requirements should be created as statements. Statements are the expressions based on the conditions of the signals. In which statement that a particular signal should be assert or de-assert should be specified in the requirements. Requirements should be defined according to the terms specified in the Requirements Standard [13]. The term "shall", for example, has the meaning of mandatory and the requirements which contains this word have to be done exactly as they defined. Therefore, the requirements must be unambiguous and complete.

The requirements should have an ID in order to be manageable and ordered. Below is an example of a requirement;

[ARINC429-RESET-12] The enable shall be set to 0 within 20 nanoseconds when reset is asserted low.

2.1.5 DO-254 Processes

In order to understand DO-254 processes, it is necessary to review ARP4754A and ARP4761 documents [14] [15]. Safety engineers determine safety requirements by performing various analyses while system engineers determine system level requirements by examining customer needs and authority requirements. All these requirements should be further studied to derive hardware level requirements. The design will be made by hardware level requirements.

The most necessary activity of the DO-254 is to adapt and organize all the company members to a work culture which is compliant to DO-254 processes. Whenever this activity occurs in an effective way, then it is easier to understand and complete the DO-254 objectives. However, if the company members do not accept and resist to this activity, compliance to the DO-254 objectives will be very difficult. After all, the cost and the time that spend will be increased.

DO-254 processes are explained in detail in the following sections. As the focus of this thesis is Verification, some of the processes are explained in detail while some of them are introduced briefly. Processes are shown in Figure 2.1 which is inspired by the DO-254 document [4].

2.1.5.1 Planning Process

Planning process is the first process of DO-254. In order to start planning process, system level activities and safety analyses should be completed, and the outputs should be provided to the planning process. The outputs of system level activities should define the system's and hardware's definition and an overview information of the relationship between hardware and the system. Moreover, safety analyses outputs should define the DAL of the functions and the components of the Hardware. These analyses contain the functional hazard assessment, preliminary system safety assessment and functional failure path analysis. The effort made for the analysis and the selection of the DAL is a critical issue. Therefore, with these information, the whole project will be shaped.

All processes should be planned comprehensively in the planning process. Definition of the design, which functions that the hardware provides for the system, how all the processes be managed, which tools are going to be used, what standards are going to be followed, which methods are going to be used, how the validation and the verification activities are going to be applied, how the configuration management and the process assurance activities are going to be applied should be decided and documented. Plan for Hardware Aspects of Certification (PHAC) document is the

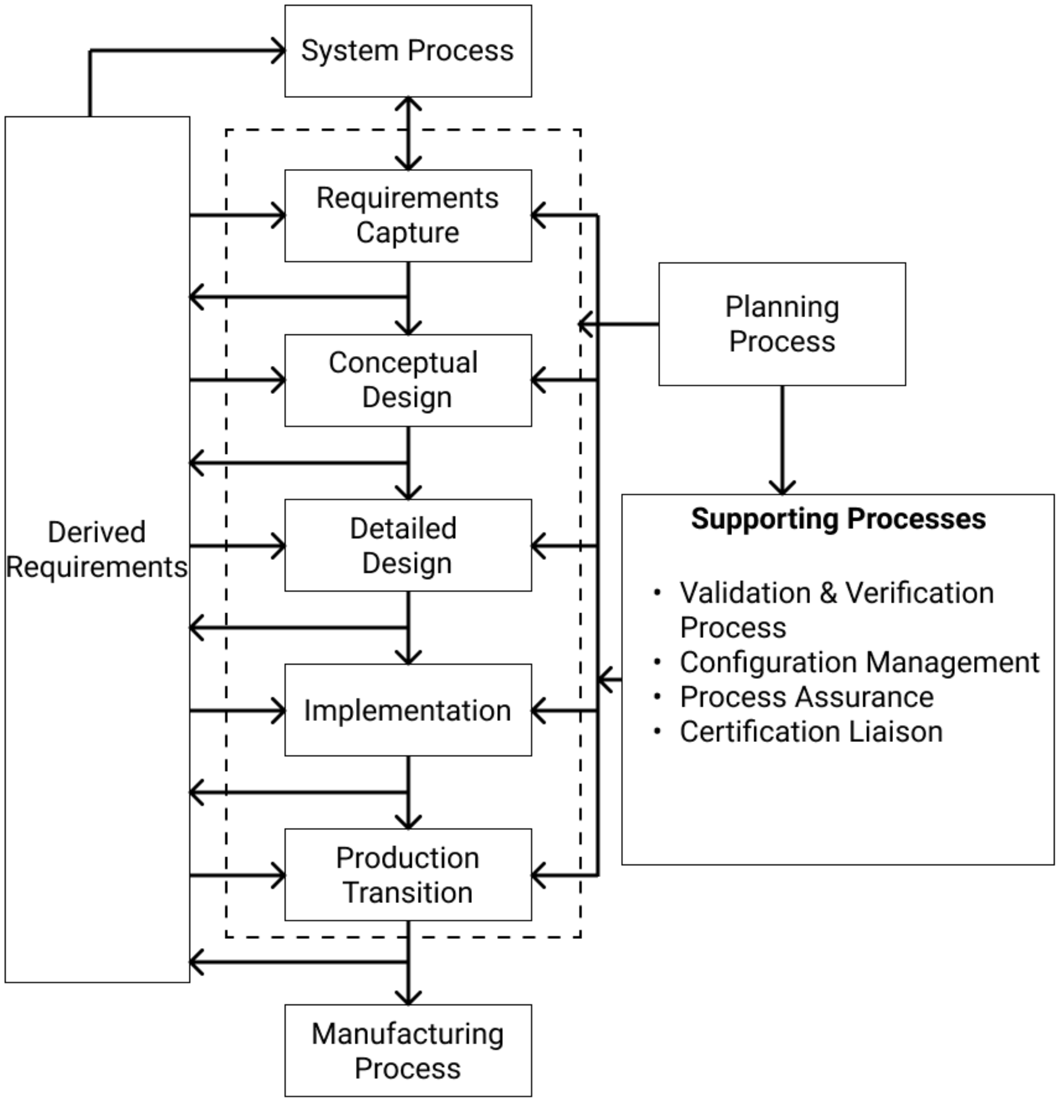


Figure 2.1 Hardware design lifecycle

main document which also referred as the “contract” between the applicant and the aviation authority which should include the above information [13]. PHAC is one of the documents which should be submitted to the aviation authority.

From the verification side, PHAC should specifically include verification team and manager information, brief definition and explanation of verification process, demonstration of relationship between the verification process and the other processes, and verification tools. The documents which are produced as outputs of the Planning Process are on below;

- Plan for Hardware Aspects of Certification
- Hardware Design Plan
- Hardware Validation Plan
- Hardware Verification Plan
- Hardware Configuration Management Plan
- Hardware Process Assurance Plan
- Requirements Standard
- Hardware Design Standard
- Validation and Verification Standard
- Archive Standard

All of the objectives for all of the plan and standard documents are specified in the 10’th section of the DO-254. For instance, PHAC document should include System Overview, Hardware Overview, Certification Considerations, Hardware Design Life cycle, Hardware Design Life Cycle Data, and Additional Considerations sections.

During the planning process, Validation and Verification plan documents are usually combined. How the requirements are validated to be correct and complete, and which verification activities are going to apply in order to cover all of the requirements should be explained in this document. Methods and activities should be selected properly in order to satisfy the objectives in sections 6.1.1 and 6.2.1 of DO-254.

In addition to the plan documents, it is necessary to prepare standard documents for designs with DAL A and B. Standards are the guidance documents that define rules and methods for a particular topic. They will be valid for all remaining processes after they are created and approved in the Planning process. Therefore, all working groups related to the standard documents should decide the decisions very carefully. Standard documents should be referenced in the relevant plan documents. For instance, the Hardware Validation and Verification Standard document should be referenced in the Hardware Validation and Verification Plan.

The Validation and Verification Standard is the standard document that includes the analysis, criteria, methods, and reviews. For validation, required topics can also be expressed in the Hardware Requirements Standard. For verification, it is the standard document that describes how the verification activities will be carried out, what is the criteria to complete the verification activities, and how the reviews should be carried out with which data.

After the planning process, SOI-1 audit will be organized between the authority and the applicant, where the authority reviews whether the planning process outputs are appropriate with DO-254.

2.1.5.2 Hardware Design Processes

Hardware design process is split into five main sub-processes as depicted in Figure 2.1.

- **Requirements Capture Process**

  Requirements Capture Process is the first process of the Hardware Design Processes. System level requirements and the Planning Process outputs are entered as inputs to this process. The main purpose of this process is to create the low level hardware requirements through these inputs. Low level hardware requirements should have direct traceability to the Low level system requirements. Entire design and the verification activities are going to be applied with the low level hardware requirements.

- **Conceptual Design Process**

  The second process is the conceptual design process, where a concept study of the design is carried out according to the lower-level requirements determined in the requirements capture process. The main purpose of this process is to demonstrate how to adapt safety and system requirements by providing high-level narration of block diagrams.

- **Detailed Design Process**

The third process is the detailed design process and it is the process where the actual design is made. For example, this is the process, where the Hardware Description Language (HDL) code is written. At the end of this process, the second audit which is SOI-2 is carried out by the authority to review whether objectives for this process are completed in accordance with DO 254.

- **Implementation Process**

  The fourth process is the implementation process and it is the process by which the completed design is transferred to the relevant environment. For example, in FPGA design, HDL code is synthesized, placement and routing are made, bitstream is created for transferring the circuit to the FPGA chip.

- **Production Transition Process**

  The last process is the production transition process and it is the last checkpoint before the production process [16]. It is ensured that the most accurate design version for production is completed, all identified errors are fixed and that there are no gaps left at the coverage of all requirements. SOI-4 audit is performed at the end of the production transition process.

2.1.5.3 Supporting Processes

Supporting processes are divided into four different groups and they are operated concurrently by all other DO-254 processes.

- **Validation & Verification Process**

  Meaning of the terms validation and verification are often confused with each other. Validation is the study in which the expectations of the customer and the aviation authority are examined to determine if they are understood correctly. This study is achieved by analyzing all of the requirements to assure that they are correct and complete. As mentioned in the Requirements section, there is no sense to verify all of the requirements perfectly if the requirements are not the appropriate ones.

  Verification, on the other hand, is to check whether all specified requirements have been met correctly. With this process, the applicant can assure to the authority that the hardware covers all of its functions and they work correctly. Because of the verification process being a supporting process, it is carried out at the particular time slots of the Design Processes and the results are expected to cover transition criteria between the processes. In the requirements capture process, lower-level requirements are evaluated separately for design and verification in order to provide independence for DAL A and B designs. The person who attends the verification activity is expected to be not involved in the design of that particular hardware.

Requirements are qualified by the verification engineers with the relevant test methods which are named as review, analysis and test. Review is the method that there is a manual or automated comparing of the hardware against particular criteria with the usage of checklists or specific documents. Analysis methods, on the other hand, are often referred to as simulation studies for PLD designs and are the most common used method to verify PLD requirements. Analysis methods are the most commonly chosen verification techniques which are required for the DAL A and B designs [17]. In the aviation industry, the automation using the computers is more preferred since the human-based method is more prone to errors. The automation can provide more reliable verification by applying constrained random verification over the UVM. Test cases are determined according to the requirements before applying the simulation. Then, testbenches are created according to these test cases. Creation of the testbench, the determination of various scenarios and their implementation to the code will be detailed in the case study section. Verification requirements determined by the requirements capture process are converted to the verification design level requirements and linked to each other. It is expected that all requirements will be met as a result of the tests which run on the device under test (DUT) with the UVM environment and constraints that determined by the requirements. There is a common approach in the industry which shapes EDA manufacturer's tool types. This approach is commonly visualized in Figure 2.2.

There should be a requirements management tool that the applicant manages all of their requirements which may be produced by EDA manufacturers. If the requirement management tool is not produced by an EDA manufacturer, there should be a tool in the middle that connects the verification results to the requirements management tool. Design tools are also connected to this tool. Verification results are the outcomes of the coverage data.

There are three coverage methods to measure whether all requirements are adequately met or not. These are functional coverage, code coverage, and assertion coverage. The entire verification process is completed when these three coverage metrics reach their target values. These methods will be defined at the further sections of this thesis. The coverage information is obtained as a report from the tools.

All files that is used for the simulation based automated verification should be

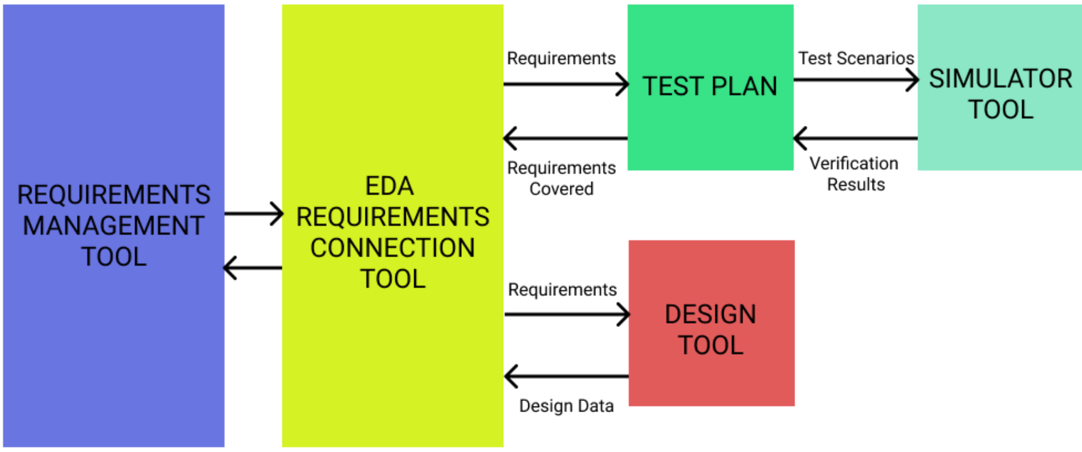


Figure 2.2 Requirements connection to design and verification

kept as the evidence of applying the verification activity. These are all for the simulation based analysis and there are more analysis after the simulation step such as Static Timing Analysis after the design implementation and in-circuit verification after uploading the bitstream on the FPGA board. There are some other analysis methods which can be used for the DO-254 verification such as formal analysis which is combination of mathematical methods for predicting the design behavior.

The test method which is the final method represents the controls made on the system when the PLD and CCA designs are finished and becomes a system. SOI-3 audit takes place at the end of the Validation & Verification process.

- **Configuration Management Process**

  Configuration Management keeps record of all the design documents along with the decisions made throughout the whole process. The main topics of Configuration Management are revision control, problem report, and change management. These topics should be settled in the planning process to keep track of all files that will be generated.

- **Process Assurance Process**

Process Assurance is for monitoring and controlling all of the objectives and activities through the whole processes. It is to ensure that future problems can be solved at the moment of appearance. Process Assurance personnel is obliged to determine whether the prepared documents comply with DO-254 through internal audits.

- **Certification Liaison Process**

  Certification Liaison personnel are obligated to contact the aviation authority and to organize audits throughout all processes.

2.2 ARINC-429 Data Bus

Before 1980s, civil aircraft equipment manufacturers developed their own bus designs. The presence of these scattered busses made the underlying system highly complex, hard to manage and maintain. Therefore, a standardized communication bus, namely ARINC-419, was developed. However, ARINC-419 has brought many shortcomings so ARINC-429 is designed to fix them. Switch and hub-like connection junctions used in the next generation of busses are not used in ARINC-429. Instead, cable connection is established as point to point. In other words, the ports are connected directly to each other.

In ARINC-429, one transmitter can send information to a maximum of 20 receivers. The data type is half duplex which means that the data goes in only one direction at a time [18]. There are two types of data communication speeds: low speed with 12.5 Kbps and high speed with 100 Kbps [19]. These two speeds cannot be used on the same ARINC-429 line, so depending on the data transmission speed requirements, one of the two speeds within the same line should be selected. Each data sent in ARINC-429 is 32 bits long and called as a word. Each word contains the following information;

1. 8 bit Label ID
2. 2 bit Source/Destination Indicator (SDI)
3. 19 bit Data
4. 2 bit Sign/Status Matrix(SSM)
5. 1 bit Parity

<table>
<tr><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr>
<tr><td>P</td><td colspan="2">SSM</td><td colspan="19">MSB DATA LSB</td><td colspan="2">SDI</td><td colspan="8">LABEL ID</td></tr>
</table>

Figure 2.3 ARINC-429 word

Unlike many data busses, enumeration of the first bit of ARINC-429 starts at 1 instead of 0. The 32nd bit represents the most significant bit and 1st bit represents the least significant bit. The entire word delivery order except the Label bits, is from the least significant bit to the most significant bit. The 8th bit is the most significant bit of the Label. Note that the Label bits are sent in the reversed order while the remaining bits are sent sequentially. The data bits are sent in the following order:

8 7 6 5 4 3 2 1 9 10 11 ..... 31 32

All equipment using the ARINC-429 data bus uses the Parity and Label bit fields in a common method in their word structure. Other fields on the other hand can be interchangeable by equipment manufacturers unless the number of bits are not exceeded. ARINC-429 uses the Bipolar Return to Zero methodology for the data transmission. Therefore, there are three values in this methodology; 1 represents +5V, Null represents 0V and -1 represents -5V. Data sent on differential binary cables is transmitted simultaneously. When sending data from channel A in a straight format, the data in channel B is reversed and sent at the same time. As an example, 10101 is sent from channel A and 01010 is sent from channel B at the same time. Since the main focus in this thesis is UVM, the digital design of ARINC-429 is utilized from an open source design [20].

2.3 Universal Verification Methodology

Today, UVM is the most commonly used PLD verification methodology all over the world. Over the years, it has been observed that there are many commonalities in code structures even though projects are different. UVM is built on a library using object-oriented programming methodology and the SystemVerilog language. The reusability is the most important advantage of the UVM which is critical for saving the project duration and budget. Thus, there will be no need to reinvent the wheel in each project. Once labor is spent using the UVM, it can be transferred to other projects.

In the UVM library, which is created for the methodology, all the tools that should be included in the structure are defined. One needs to set up their own verification environment using these tools. Each of the elements in the UVM is used for a specific purpose. At the same time, component communication network in the verification environment is provided automatically by the UVM. In this way, the verification team can devote their time to the most important task, the delivery of input data and the observation of the output data.

UVM and its structure will be detailed in the Case Study section of this thesis. DO-254 processes are requirement-based. Each process is managed upon proving that all requirements are met. For a process to be considered complete, its requirements must be correct and complete, and then fully met. The participant is responsible for determining the methods by which verification requirements will be met. The most important thing for the authority is that all requirements are met in accordance with DO-254.

EDA Playgrounds [21], which is a free platform including the commonly used simulation tools, has been widely used for academic purposes. This paper, for example, uses EDA Playgrounds for the experiments [22]. EDA Playgrounds is used for the whole development of the Case Study and the coverage reports are achieved via the usage of the academic licensed tools from TÜBİTAK BİLGEM TUTEL's [23] laboratory environment.

3
CASE STUDY: VERIFICATION OF ARINC-429

3.1 Loopback IP Core

Within the scope of this thesis, it is aimed to verify that transmitter (TX) and receiver (RX) modules in the ARINC-429 design are working correctly. TX and RX modules are connected to each other as "loopback" as shown in Figure 3.1. For the sake of simplicity, ARINC-429 Loopback IP Core does not include a CPU interconnection. The signals are generated and transmitted to the TX module in accordance with the protocol rules, and as a result, it is checked whether the correct data is received from the RX module.

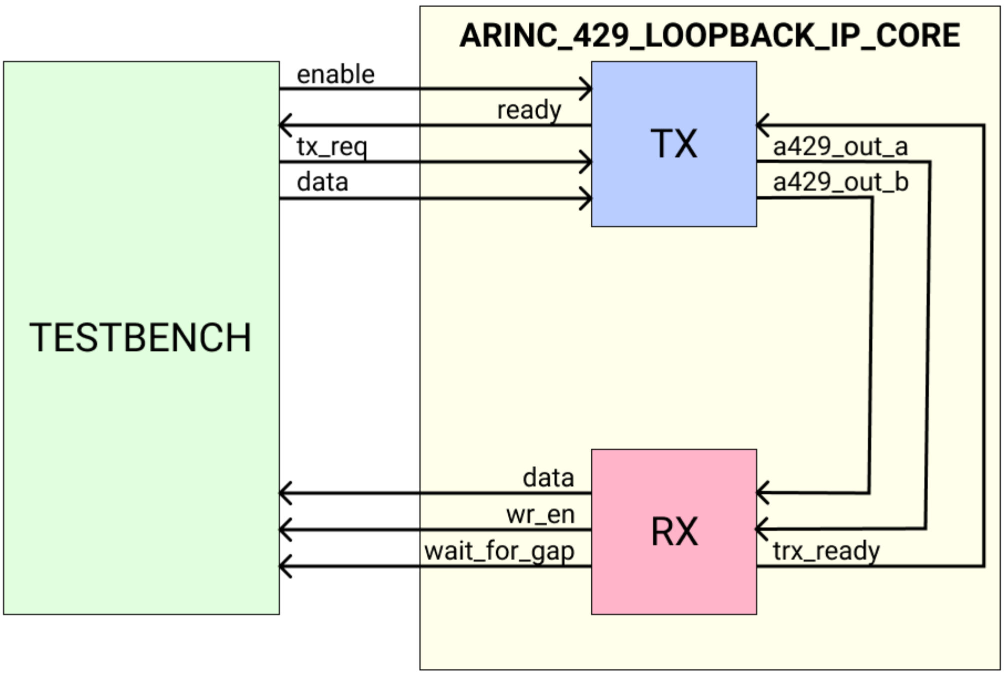


Figure 3.1 ARINC-429 loopback IP core

The "enable" signal must be high for data communication to be started. Therefore, the testbench sets the "enable" signal to high. After the TX module detects the "enable" signal as high, it waits for the “trx_ready” signal to be high. The RX module sets “trx_ready” signal to high whenever it is ready to receive a data. When both “enable” and “trx_ready” signal becomes high, TX module sets the "ready" signal to high indicating that it is ready to receive data from the testbench. After that, the testbench sets "tx_req" signal to high and sends the ARINC-429 WORD data. In this way, the handshake takes place, and the ARINC-429 WORD is safely transmitted to the TX module.

The data transmission from the TX module to the RX module is carried out through two separate channels which are named as a and b. Data is sent simultaneously through channel a and channel b, where the channel b’s data is the reversed version of the channel a’s data. This simultaneous data transmission is used to verify that the data is transferred properly.

The RX module performs a parity check after all data is received. In this way, it is tested whether the data is valid according to the parity rule of ARINC-429 protocol. If the data is valid, the RX module sets the "wr_en" signal to high and transmits the data to the testbench. The testbench checks if the "wr_en" signal is high or low in order to understand whether the data is valid or not. The testbench also waits for the “wait_for_gap” signal to be high in order to send a next data to the TX module.

3.2 Testbench Implementation with UVM

In this thesis, it is aimed to demonstrate the verification study of ARINC-429 design on a sample scenario with the usage of the UVM. A certain number of random ARINC-429 WORDs is generated in accordance with the rules of the protocol and sent sequentially. The generated WORDS are first transmitted from the testbench to the TX module and the testbench expects receiving correct WORDs from the RX module since the TX and RX modules are connected to each other as "Loopback" on the design side. UVM is used for creating the testbench. The features and components of the UVM are described briefly in the subsequent sections below.

3.2.1 UVM Features

Software languages had many advantageous features that digital design languages did not have. SystemVerilog Design and Verification Language is created for this issue and these features were implemented to the SystemVerilog. The most important one of these features was the Object Oriented Programming methodology. Verification of the various digital designs includes very similar activities. With the power of Object Oriented Methodology, reusability is maximized which means that testbenches and their components for a particular design were transferred to the new designs. Therefore, many companies and EDA tool manufacturers have started to use internal standard verification environments over the years. However, the fact that these environments were not accessible to everyone and they all had their good features as well as their shortcomings created the need for a new and centralized standard. Leading EDA tool manufacturers gathered these verification environments and components and created the UVM which is now an IEEE standard.

In order to use the UVM, it is necessary to download the UVM library [2] from Accelera's site and use a suitable simulation tool which has the capability to work with the UVM. UVM library consists of various classes and these classes have a hierarchical structure. UVM classes are shown in Figure 3.2 that is inspired by the ChipVerify's website [24].

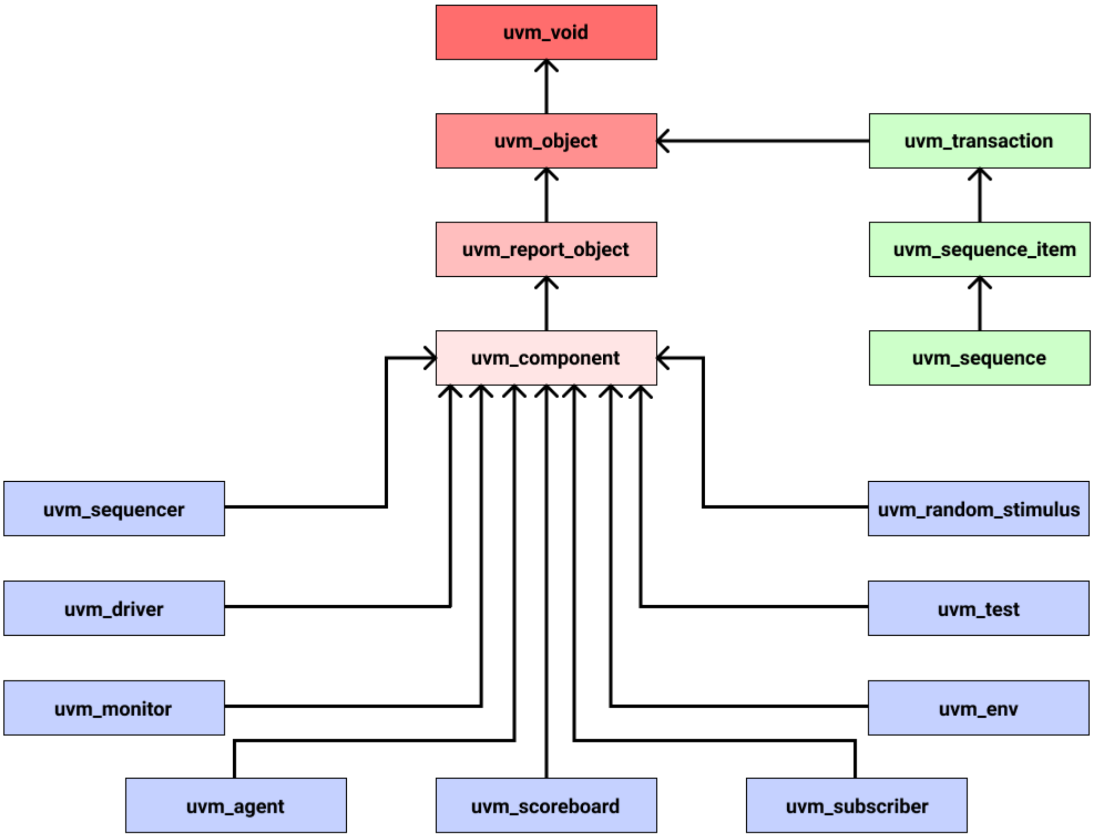


Figure 3.2 UVM base classes

Various types of classes are shown in the Figure 3.2. The one that use the UVM for the verification should inherit these classes and create their classes upon them in order to use the UVM methodology and the features it contains. The most fundamental class of the UVM is the uvm_component class, where the entire verification architecture is created. There are various clases under the uvm_component class that is used for different purposes which improves reusability.

There are uvm_object classes which have different purposes from the component classes. These classes are used to create the data objects that is used by the component classes for the data transmission. For example, the uvm_sequence_item class creates signals which will be sent to the design from the testbench and also it is used to observe the data from the design. Constrained Random Verification is implemented in this class by determining the constraints according to the design protocol with the usage of “constraint” structure of the SystemVerilog language. Certain signals and variables included in each object of this class type are created by the simulator according to the determined constraints. The Figure 3.3 below is the code snippet that shows how the 32-bit variable named as the "data_to_TX" of the class "transaction_a429" that is inherited from the uvm_sequence_item in the ARINC-429 Testbench is created with the usage of the constraints according to the protocol.

SV/Verilog Testbench

```
class transaction_a429 extends uvm_sequence_item;
  rand bit  [32:1]  data_to_TX;
  bit       [32:1]  data_from_RX;
  bit               par_en;
  bit               ready;
  bit               tx_req;
  bit               wait_for_gap;

  //CONSTRAINTS

  constraint c1 {data_to_TX[32] == 1'b1;};
  constraint c2
  {
    (data_to_TX[1] ^ data_to_TX[2] ^ data_to_TX[3] ^ data_to_TX[4] ^
data_to_TX[5] ^ data_to_TX[6] ^ data_to_TX[7] ^ data_to_TX[8] ^ data_to_TX[9] ^
data_to_TX[10] ^ data_to_TX[11] ^ data_to_TX[12] ^ data_to_TX[13] ^
data_to_TX[14] ^ data_to_TX[15] ^ data_to_TX[16] ^ data_to_TX[17] ^
data_to_TX[18] ^ data_to_TX[19] ^ data_to_TX[20] ^ data_to_TX[21] ^
data_to_TX[22] ^ data_to_TX[23] ^ data_to_TX[24] ^ data_to_TX[25] ^
data_to_TX[26] ^ data_to_TX[27] ^ data_to_TX[28] ^ data_to_TX[29] ^
data_to_TX[30] ^ data_to_TX[31] ^ data_to_TX[32]) == 1;
  };
```

Figure 3.3 Constraints of the sequence item

In the class, first, the "data_to_TX" variable is created by using the "rand" keyword which is a SystemVerilog feature that randomizes the determined signals. Afterwards, constraints are determined so that the simulator will randomize the signal in accordance with the design protocol. In the ARINC-429 protocol, the value of the 32nd bit which is named as "parity", must be 1 in order for the data to be called as a valid data. Also, the sum of the numbers of all bits must be an odd number.

UVM library includes wide variety of useful tools. For example, TLM ports are classes that is used as objects inside component classes, providing components to exchange data with each other. Ports facilitate the transmission of data with a standard structure with the methods they contain.

Another feature that UVM classes have is the UVM Phases. With the usage of the phase methods in component classes, all operations are separated into various phases before, during and after the simulation and take place sequentially. In the “build” phase all components are created. In the “connect” phase, the connection between all components is made. The “run” phase is the phase where the simulation takes place. At last, the “final” phase is performed and the simulation is ended.

As all these advantageous features are already available in the UVM library and the user directly receives and uses these features, the user gains great advantages in terms of time and reusability. In addition to that, since UVM is an IEEE standard and is used by the whole industry, intelligibility between companies increases significantly.

3.2.2 Testbench Architecture

UVM classes of the testbench are shown in Figure 3.4 and their description is provided in the subsequent sections below.

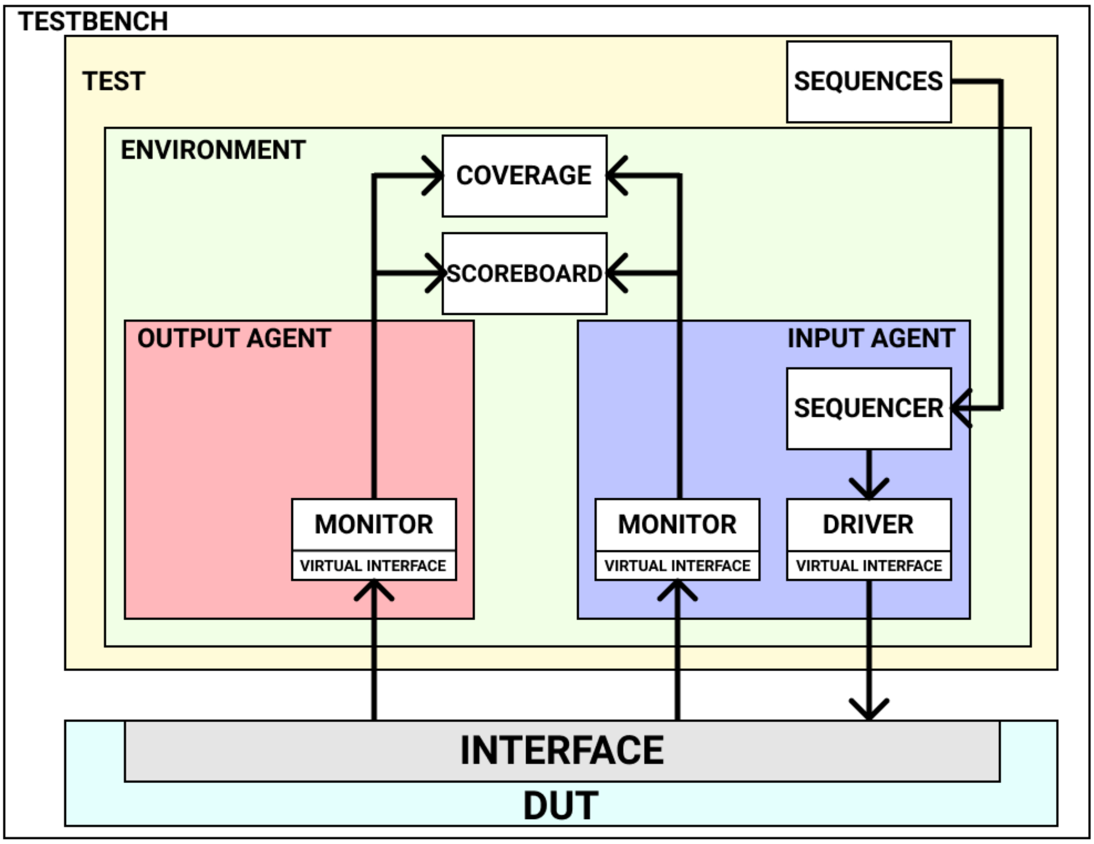


Figure 3.4 UVM testbench architecture

3.2.2.1 Sequence & Sequencer

The Sequence class is the class in which the scenarios in testbench are created. It extends the uvm_sequence class. It creates and orders the Sequence Items and transmits them to the Sequencer. Sequencer class provides data transmission between the Sequence and the Driver with the usage of built-in FIFO structures that it contains. It extends the uvm_sequencer class. 1024 ARINC-429 Words were created by the Sequence and sent to the Sequencer in order in the Testbench.

3.2.2.2 Interface

Before explaining the other UVM classes, I think it is necessary to mention the Interface. It is the SystemVerilog structure that enables data transmission between the design and the testbench. Signal level connection between the design and the testbench is made by the Interface. At the same time, Assertions that monitor the necessary conditions of the signals during the entire simulation are also created within the Interface. Assertions will be explained in future sections.

3.2.2.3 Driver

Driver transmits the relevant data that received from the Sequencer to the design over the Interface in accordance with the protocol's timing rules. It extends the uvm_driver class. As shown in Figure 3.4, Driver transmits the data from the Testbench to the TX module of the design.

In the ARINC-429 Driver class, first of all the "enable" signal is set to high. Driver waits for the "ready" signal to be set high by the design. After the "ready" signal is detected as high, the "tx_req" signal is set to high and transmitted to the design via the interface. At the same time, the "data_to_TX" that pulled from the Sequencer is transmitted to the design via the Interface. Driver sets the "tx_req" signal to low whenever "ready" is low.

3.2.2.4 Monitor

Monitor class allows us to observe the data which is sent to the design and the data which is received from the design. It extends the uvm_monitor class. In both observation states, both monitors send the received data to the Scoreboard and the Coverage for the comparison and functional coverage purposes. In the Testbench scenario, the monitor in the INPUT AGENT observes the data, which is sent to the TX module. On the other hand, the monitor in the OUTPUT AGENT observes the data which is received from the RX module.

The monitor class in the INPUT AGENT waits for the “tx_req” signal to be set to high by the Driver. If the signal is high, it observes all of the interface signals and sends them to the Scoreboard and Coverage through the TLM Ports.

The monitor class in the OUTPUT AGENT waits for two signals to be set to high by the RX Module separately. These signals are named as “wait_for_gap” and “par_en”. Monitor observes all of the interface signals with these signals being high and sends them to the Scoreboard and Coverage through the TLM Ports.

3.2.2.5 Agent

Agent class covers the components that perform data transmission and reading. It extends the uvm_agent class. Agents are responsible for integrating the fundamental components together. Agents which include Sequencer and Driver are called as active agents because they transmit the data to the design. Active Agents are also connect the Sequencer to the Driver. On the other hand, Agents which only include Monitor for observing the design are called as passive Agents.

3.2.2.6 Scoreboard

Scoreboard class compares observed data from the Monitor inside the INPUT Agent and the Monitor inside the OUTPUT Agent as described in the Monitor section. It extends the uvm_scoreboard class. Data which received from both Monitors are expected to be matched. Monitors write their data to the Scoreboard at different simulation times. In order to match the appropriate data, TLM FIFO ports are used in the Scoreboard.

3.2.2.7 Coverage

Coverage class provides the functional coverage. It extends the uvm_subscriber class. uvm_subscriber class has the advantage of built in UVM Analysis ports which normally be declared manually on the other classes. In this class, covergroup is defined for the functional coverage. Functional coverage will be explained at the further sections.

3.2.2.8 Environment

Environment class covers the Coverage, Scoreboard and all the Agents, and also establishes the necessary connections between them with the usage of the TLM ports. It extends the uvm_env class.

3.2.2.9 Test

Test class covers the Sequence and Environment. It extends the uvm_test class. The name of the Test class is important because it will be used at the Testbench module's run_test method. Test class also connects Sequence to the Sequencer.

3.2.2.10 Testbench

Testbench is the module that includes general steps for starting the whole verification activity that are defined on below:

- All the necessary UVM files and all of the classes are imported via a package.
- Interface and DUT are instantiated.
- Connection between the Interface and DUT is made.
- Interface is added to the UVM database.
- Clock and reset are created.
- UVM test is run.

3.3 Assertions

Assertions have been used for a long time in the verification activities of the digital designs. An assertion is a SystemVerilog language structure that monitors the design and checks whether a specific part of it behaves correctly. If design behaves correctly, assertions give a "pass" feedback to the simulator. On the other hand, the simulator will give an error if the design behaves wrongly. Since the error happens because of the assertion which monitors a specific behavior of the design, it becomes much easier to find error's source. One of the most important reasons for using Assertions is the ease of finding the cause and source of the error. If the assertions are not used, it would be necessary to search from the outermost of the design to the innermost for finding the source of the error as in Figure 3.5. With the usage of the assertions, the source of the error can be found easily as shown in Figure 3.6. From this point of view, the verification process becomes significantly easier with the usage of the assertions when errors are founded at the early stage of the design process which contributes whole processes in terms of time.

Another important reason to use the assertions is that the design requirements actually specify how the design should behave which means that assertions are directly cover

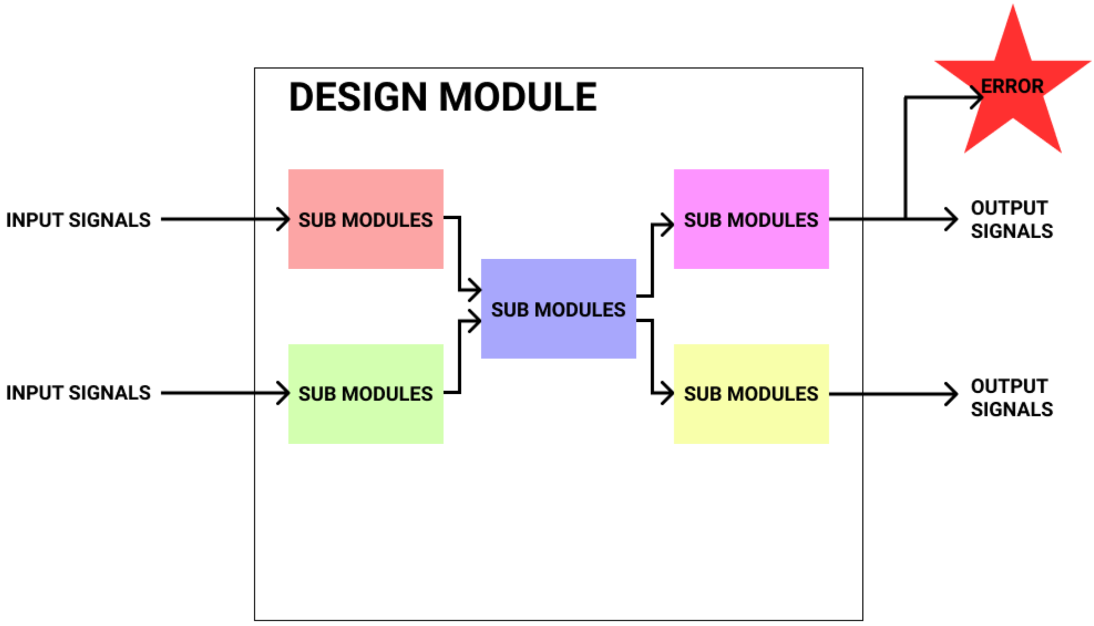


Figure 3.5 Debugging without assertions

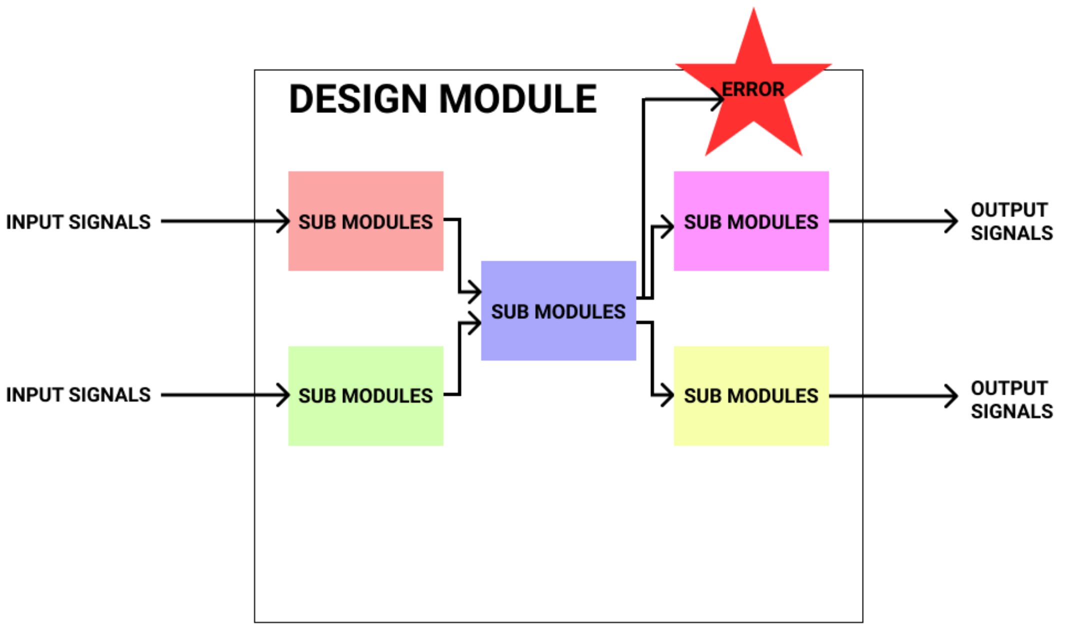


Figure 3.6 Debugging with assertions

the requirements. If all of the requirements are correct and complete, we can ensure that our design works correctly if we specify each of these requirements as assertions and display that all assertions pass in all test scenarios and conditions.

These reasons show why assertion is such a powerful verification method. In addition to what has been described, assertions are the structures that are designed completely independent from the design. They are usually created only by the usage of the input-output signals of the design and they do not manipulate the design. While the design teams create the design with the determined requirements, the verification teams create the assertions independently which leads verification process to be in much higher quality. Ambiguities in the requirements will be prevented and better requirements will be shaped as a result. Thus, it can be understood that the assertions also contribute to the validation of the requirements. As I explained in the Validation and Verification section, the verification of the design is completed when the assertion coverage, functional coverage and code coverage percentages are hit to the aimed value. These coverage data are the proof that a design works exactly as described in the requirements.

In the Use Case study, firstly, the requirements were determined in order to verify the ARINC-429 Loopback IP Core. The required assertions are written to cover all of the requirements along with the UVM testbench. How SystemVerilog assertions are designed is beyond the scope of this thesis. The requirements are shown on the Table 3.1.

Figure 3.7 shows an example assertion which is implemented for the [ARINC429-ready-1] requirement.

```
property ready_1;
    @(posedge clk) disable iff (reset)
  (!$isunknown(data_from_RX)) && ($fell(wait_for_gap)) |=> ##[0:$] ($rose(ready));
endproperty: ready_1

A_ready_1: assert property (ready_1);
C_ready_1: cover property (ready_1);
```

Figure 3.7 [ARINC429-ready-1]

Other assertions which are corresponding to the other requirements are shown in the APPENDIX B.

Table 3.1 Requirements

Req. ID	Requirement
TRANSITION REQUIREMENTS	
[ARINC429-ready-1]	ready signal shall be asserted after the wait_for_gap signal being deasserted.
[ARINC429-ready-2]	ready shall not be asserted if the enable is low.
[ARINC429-tx_req-1]	tx_req shall be asserted in the same clock cycle after the ready being asserted.
[ARINC429-tx_req-2]	tx_req shall not be asserted if the ready is low.
[ARINC429-data_to_TX-1]	data_to_TX shall be start sending data in the same clock cycle after the tx_ req being asserted and data_to_TX shall not float until the tx_ req being asserted again.
[ARINC429-wr_en-1]	wr_en shall be asserted only if the data is valid.
[ARINC429-data_from_RX-1]	data_from_RX shall be start sending data in the same clock cycle after the wr_en being asserted.
RESET REQUIREMENTS	
[ARINC429-reset-1]	After reset is asserted ready, tx_req, data_to_TX, data_from_RX and wr_en shall be valued by 0.

3.4 Functional Coverage

Functional coverage is the metric that measures how much of the functions in the design are covered. Assertions can also be counted as a part of the functional coverage if we consider requirements as functions, but assertions are considered as a separate metric in this thesis. For this reason, functional coverage is often referred to as the metric that covers which bits in the entire bit range of the signals and variables in the design are exercised during the simulation. It is especially useful for signals and variables created with the constrained random verification. Because it is very important for a participant to prove that they provide adequate coverage data to the aviation authority that they exercised the adequate number of bits inside the signals and variables that are coherent to the design specifications. Note that the covergroups are the SystemVerilog structures for the functional coverage. How the covergroups are designed is beyond the scope of this thesis.

In the Use Case study, the created covergroup is shown in Figure 3.8 below;

```
covergroup a429_coverage;
  option.per_instance = 1;
  option.comment = "Counter Coverage";
  PARITY_ENABLE:  coverpoint coverage_tr.par_en;
  READY:          coverpoint coverage_tr.ready;
  TX_REQ:         coverpoint coverage_tr.tx_req;
  WFG:            coverpoint coverage_tr.wait_for_gap;
endgroup: a429_coverage
```

Figure 3.8 Covergroup

The data come from the monitors at various relevant times of the simulation that includes the values of the "par_en", "ready", "tx_req" and "wait_for_gap" signals information are covered in the the "a429_coverage" covergroup.

3.5 Code Coverage

Code coverage is the coverage metric that measures the percentage of the lines executed on the code. Tools provide a wide variety of code coverage metrics. Types of the code coverage metrics [25] and their descriptions are on below:

Statement Coverage / Line Coverage: It is a metric that measures whether all lines and expressions in the entire code are executed.

Block Coverage: Begin-end, if-else, case, for and so forth are called as blocks. The block coverage metric is a metric that measures whether blocks are executed.

Branch Coverage: Conditions like if-else are called as branches. Branch coverage is the metric that measures all conditional probabilities of a conditional statement.

Expression Coverage: This is the metric that measures the output possibilities of the logical statements and covers which of them are executed.

State/Transition Coverage: It is the metric that measures whether all states in state machine are executed and all of the state's transition to the other states are happened.

Toggle Coverage: It is the metric that provides the measurement of the change on the values of all signals and ports. Thanks to this metric, unused or steady signals can be easily detected.

As a result of code coverage, it is checked whether the code is executed. However, the execution of the code does not mean that the code is functionally correct. Therefore, code coverage is a method that must be applied, but it is never sufficient on its own. This is exactly why assertion and functional coverage should be implemented alongside code coverage.

4
RESULTS AND DISCUSSION

In the verification scenario, 1024 WORDs were sent consecutively from Testbench to ARINC-429 Loopback IP. It is aimed to observe the same WORD on each from ARINC-429 Loopback IP in return. It is also expected that the specified requirements will be met during the simulation. As explained in the Verification section, assertion coverage, functional coverage and code coverage metrics are expected to be at the aimed percentage. The aimed percentage for functional coverage and assertion coverage is %100. For code coverage, the determined percentage is above %90. Why the expected percentage for code coverage is not %100 will be explained in the code coverage results section. In order to observe the reset state during the simulation, 100 reset signals are asserted temporarily and at different times during the data transmission. Waveform of the first WORD's data transmission is shown in Figure 4.1.

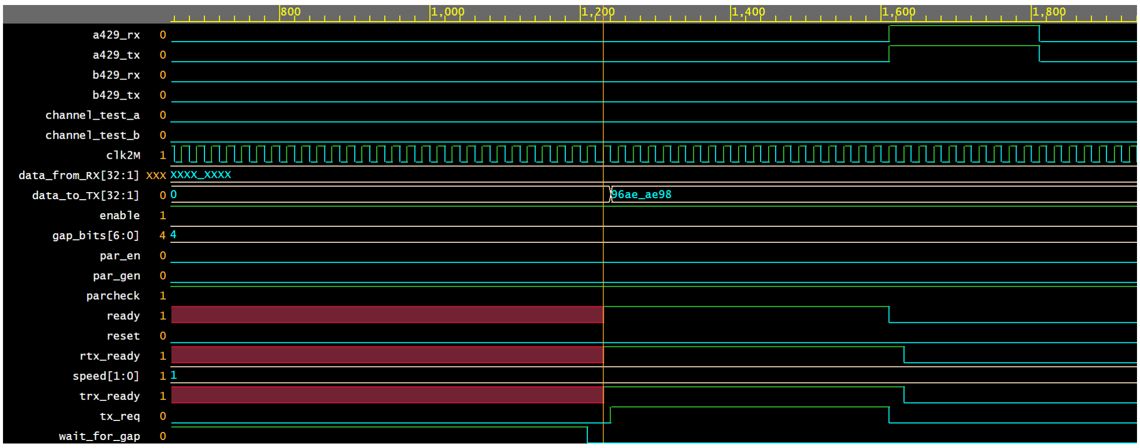


Figure 4.1 First WORD's data transmission

A portion of the waveform is shown in Figure 4.2.

As can be seen in Figure 4.1, first the "enable" signal is asserted by the Testbench and the "ready" signal is waited to be asserted from the Tx module. On the other hand, TX checks the "trx_ready" signal to be asserted by the RX module meaning that the RX is ready to receive data. After seeing the "trx_ready" as asserted, TX module

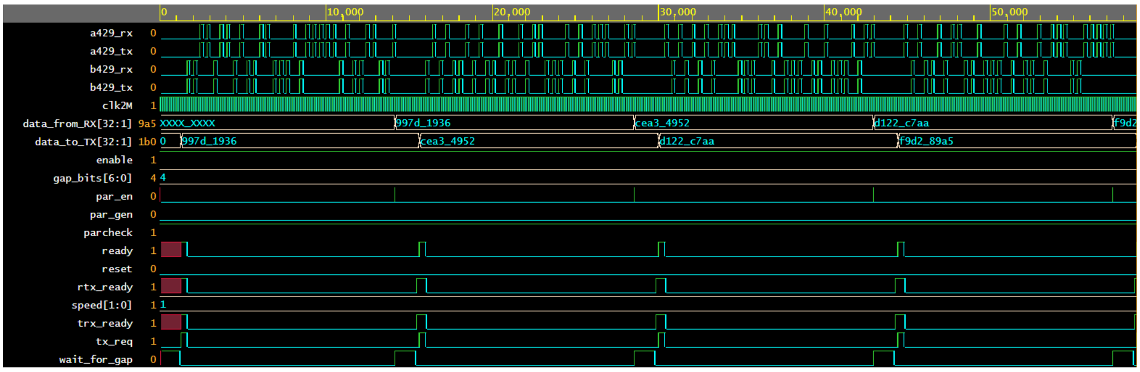


Figure 4.2 A snippet from the waveform

asserted the "ready" signal. Afterwards, Testbench asserted the "tx_req" signal and sent the ARINC-429 WORD to the TX module with the "data_to_TX" in parallel at once. Then the TX module sent the data serially to the RX module. At the last bit of the transmission, the RX module checked the Parity bit and showed that the data was correct by asserting the "parity" signal. Then the RX module asserted the "wait_for_gap" signal to indicate that it is waiting for the selected gap time in the ARINC-429 protocol. Testbench waited for the "wait_for_gap" signal to be de-asserted before sending the new WORD to the ARINC-429 Loopback IP. The results from the Scoreboard, which compares the sent and received WORDs in the WORD transmission explained above, are shown in Figure 4.3.

```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test random_TX_test_a429...
# UVM_INFO scoreboard_a429.svh(58) @ 14150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# UVM_INFO scoreboard_a429.svh(59) @ 14150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = 997d1936 matched with expected data = 997d1936
# UVM_INFO scoreboard_a429.svh(58) @ 28550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# UVM_INFO scoreboard_a429.svh(59) @ 28550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = cea34952 matched with expected data = cea34952
# UVM_INFO scoreboard_a429.svh(58) @ 42950: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# UVM_INFO scoreboard_a429.svh(59) @ 42950: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = d122c7aa matched with expected data = d122c7aa
# UVM_INFO scoreboard_a429.svh(58) @ 57350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# UVM_INFO scoreboard_a429.svh(59) @ 57350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = f9d289a5 matched with expected data = f9d289a5
# UVM_INFO scoreboard_a429.svh(58) @ 71750: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
```

Figure 4.3 Comparison of the WORDs from the Scoreboard

All the WORDs were correctly transmitted to the TX module and the WORDs observed from the RX module matched to them except for the 100 reset conditions. Reset condition can be seen in Figure 4.4.

It can be seen in Figure 4.4 that the TX module was transmitting WORD to the RX module before the reset condition occurred. However, with the "reset" signal being asserted, the value of the data of the TX and RX modules became 0. The data transmission continued after the reset condition. Scoreboard results of the reset condition are shown in Figure 4.5.

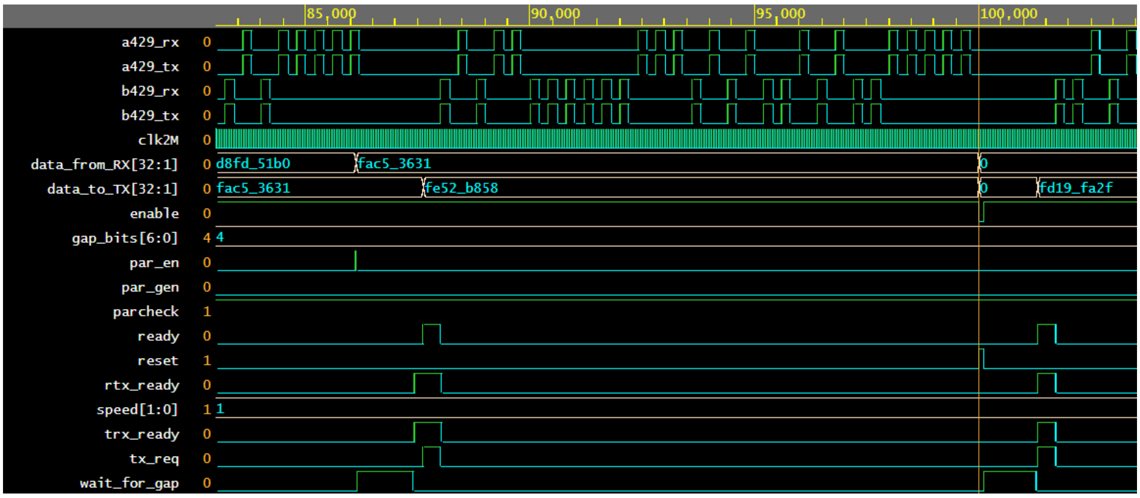


Figure 4.4 Reset condition

```
# UVM_INFO driver_a429.svh(62) @ 100000: uvm_test_top.base_test_environment.environment_active_agent.agent_driver [driver_a429] Resetting Signals
# UVM_ERROR scoreboard_a429.svh(62) @ 100110: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data 0 does not match expected fe52b858
```

Figure 4.5 Scoreboard reset condition

4.1 Coverage

Overall coverage score is shown in Figure 4.6.

Exclusion Rule Type	UNR	Name	Overall Average Grade	Overall Covered	Assertion Status Grade
		Verification Metrics	96.5%	1062 / 1270 (83.62%)	100%

Figure 4.6 Overall coverage score

The percentage is the mixed final result of the assertion coverage, functional coverage and the code coverage. These coverage metrics are explained in the following sections.

4.1.1 Assertion Coverage

As shown in Figure 4.6, assertions are %100 covered. The details of the assertion coverage are shown in Figure 4.7.

4.1.2 Functional Coverage

The signals in the covergroup, which is the indicator of the functional coverage metric, are %100 covered as shown in Figure 4.8.

Name	Overall Average Grade	Assertion Status Grade	Enclosing Entity	Source Code
C_reset_1	100%	n/a	top_a429.if_a429	C_reset_1: cover property (reset_1); //$display($stime,"reset_1 PASSED");
A_reset_1	100%	100%	top_a429.if_a429	A_reset_1: assert property (reset_1);
C_data_from_RX_1	100%	n/a	top_a429.if_a429	C_data_from_RX_1: cover property (data_from_RX_1); //$display($stime,"data_from_RX_1 PASSED");
A_data_from_RX_1	100%	100%	top_a429.if_a429	A_data_from_RX_1: assert property (data_from_RX_1);
C_wr_en_1	100%	n/a	top_a429.if_a429	C_wr_en_1: cover property (wr_en_1); //$display($stime,"wr_en_1 PASSED");
A_wr_en_1	100%	100%	top_a429.if_a429	A_wr_en_1: assert property (wr_en_1);
C_data_to_TX_1	100%	n/a	top_a429.if_a429	C_data_to_TX_1: cover property (data_to_TX_1); //$display($stime,"data_to_TX_1 PASSED");
A_data_to_TX_1	100%	100%	top_a429.if_a429	A_data_to_TX_1: assert property (data_to_TX_1);
C_tx_req_2	100%	n/a	top_a429.if_a429	C_tx_req_2: cover property (tx_req_2); //$display($stime,"tx_req_2 PASSED");
A_tx_req_2	100%	100%	top_a429.if_a429	A_tx_req_2: assert property (tx_req_2);
C_tx_req_1	100%	n/a	top_a429.if_a429	C_tx_req_1: cover property (tx_req_1); //$display($stime,"tx_req_1 PASSED");
A_tx_req_1	100%	100%	top_a429.if_a429	A_tx_req_1: assert property (tx_req_1);
C_ready_2	100%	n/a	top_a429.if_a429	C_ready_2: cover property (ready_2); //$display($stime,"ready_2 PASSED");*/
A_ready_2	100%	100%	top_a429.if_a429	A_ready_2: assert property (ready_2);
C_ready_1	100%	n/a	top_a429.if_a429	C_ready_1: cover property (ready_1); //$display($stime,"ready_1 PASSED");
A_ready_1	100%	100%	top_a429.if_a429	A_ready_1: assert property (ready_1);

Figure 4.7 Assertion coverage score

Block Expression Toggle Statement FSM **Cover Groups** Assertions

Cover Groups Filter by name: Filter... ×

Exclusion Rule Type	UNR	Name	Overall Average Grade	Overall Covered	Enclosing Entity	Source Code
		coverage_a429::a429_coverage	100%	8 / 8 (100%)	package_a429	covergroup a429_coverage;

Items Filter by name: Filter... ×

Exclusion Rule Type	UNR	Name	At Least	Overall Average Grade	Overall Covered	Source Code
		PARITY_ENABLE	1	100%	2 / 2 (100%)	PARITY_ENABLE: coverpoint coverage_tr.par_en;
		READY	1	100%	2 / 2 (100%)	READY: coverpoint coverage_tr.ready;
		TX_REQ	1	100%	2 / 2 (100%)	TX_REQ: coverpoint coverage_tr.tx_req;
		WFG	1	100%	2 / 2 (100%)	WFG: coverpoint coverage_tr.wait_for_gap;

Bins Filter by name: Filter... ×

Exclusion Rule Type	UNR	Name	Overall Average Grade	Overall Covered	Score	At Least	Source
		auto[0]	100%	1 / 1 (100%)	2047	1	PARITY_ coverpoi coverage
		auto[1]	100%	1 / 1 (100%)	923	1	PARITY_ coverpoi coverage

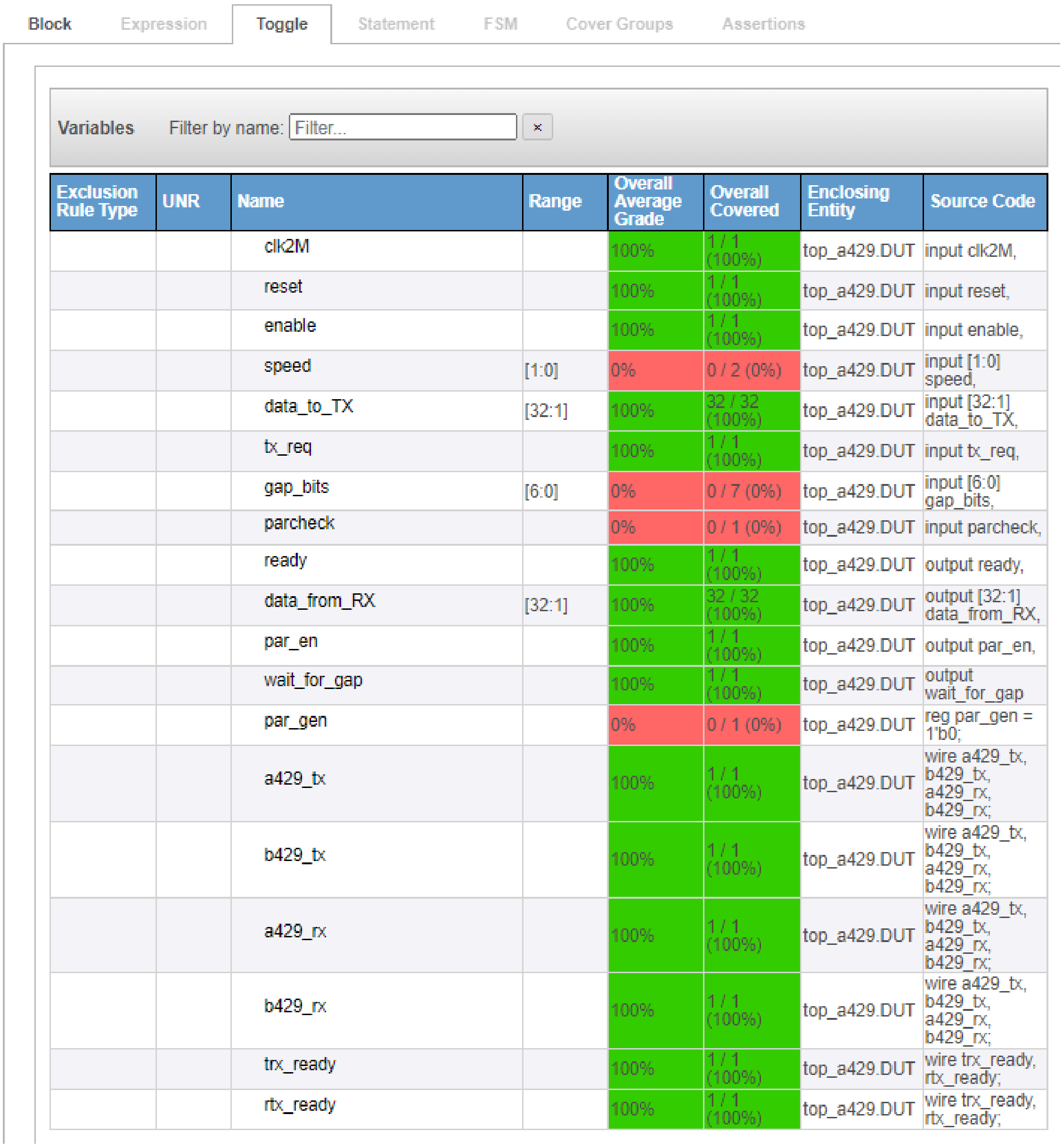
Figure 4.8 Functional coverage score

As an example, the percentage of the coverpoint named PARITY_ENABLE, which is the "par_en" signal's coverpoint, is shown in detail in Figure 4.8.

4.1.3 Code Coverage

The reason why the overall coverage that appears in Figure 4.6 is not %100 is because some parameters in the code coverage are not executed. For example, as described in the ARINC-429 section, there are two speed types in the ARINC-429 protocol. In the verification scenario, the simulation was carried out by selecting 100 Kbps speed mode. Since the variable “speed”, is the variable for the speed conditions in the design code, it always stays at the same value and reduces the percentage of code coverage. Figure 4.9 shows other constant variables like this example.

For a complete verification activity, all parameters in the design code must be ensured to be executed. However, since this thesis was carried out as an academic study, a simple configuration of the ARINC-429 design is selected and the verification is carried out over this configuration.



Block Expression Toggle Statement FSM Cover Groups Assertions

Variables Filter by name: Filter... ×

Exclusion Rule Type	UNR	Name	Range	Overall Average Grade	Overall Covered	Enclosing Entity	Source Code
		clk2M		100%	1 / 1 (100%)	top_a429.DUT	input clk2M,
		reset		100%	1 / 1 (100%)	top_a429.DUT	input reset,
		enable		100%	1 / 1 (100%)	top_a429.DUT	input enable,
		speed	[1:0]	0%	0 / 2 (0%)	top_a429.DUT	input [1:0] speed,
		data_to_TX	[32:1]	100%	32 / 32 (100%)	top_a429.DUT	input [32:1] data_to_TX,
		tx_req		100%	1 / 1 (100%)	top_a429.DUT	input tx_req,
		gap_bits	[6:0]	0%	0 / 7 (0%)	top_a429.DUT	input [6:0] gap_bits,
		parcheck		0%	0 / 1 (0%)	top_a429.DUT	input parcheck,
		ready		100%	1 / 1 (100%)	top_a429.DUT	output ready,
		data_from_RX	[32:1]	100%	32 / 32 (100%)	top_a429.DUT	output [32:1] data_from_RX,
		par_en		100%	1 / 1 (100%)	top_a429.DUT	output par_en,
		wait_for_gap		100%	1 / 1 (100%)	top_a429.DUT	output wait_for_gap
		par_gen		0%	0 / 1 (0%)	top_a429.DUT	reg par_gen = 1'b0;
		a429_tx		100%	1 / 1 (100%)	top_a429.DUT	wire a429_tx, b429_tx, a429_rx, b429_rx;
		b429_tx		100%	1 / 1 (100%)	top_a429.DUT	wire a429_tx, b429_tx, a429_rx, b429_rx;
		a429_rx		100%	1 / 1 (100%)	top_a429.DUT	wire a429_tx, b429_tx, a429_rx, b429_rx;
		b429_rx		100%	1 / 1 (100%)	top_a429.DUT	wire a429_tx, b429_tx, a429_rx, b429_rx;
		trx_ready		100%	1 / 1 (100%)	top_a429.DUT	wire trx_ready, rtx_ready;
		rtx_ready		100%	1 / 1 (100%)	top_a429.DUT	wire trx_ready, rtx_ready;

Figure 4.9 Toggle coverage

4.2 Additional Verification Activities

4.2.1 Channels of ARINC-429

Verification activites are performed on the simulation environment, where data transmission are applied in a perfect condition. In the real-world scenarios, there may be data errors because of cable distances. There are some additional verification activities made to simulate the real world problems.

There are two separate channels in the ARINC-429 protocol which prevents wrong data transmission by checking that both channels should have opposite values at the same time during the transmission as it is discussed in Section 2.2. Although the probability of the erroneous data transmission is low, long cables used in the aircraft may still cause faulty data transmission.

In the course of the WORD transmission simulated with the UVM, 5 WORD's channel B data are changed to its opposite value during their transmission. At these moments it is observed that channel A's data is equal with channel B's data. This moment is shown in Figure 4.10. It is observed that RX drops the data because it is illegal to the protocol that both channels should be valued opposite to each other at the same time and RX remain unchanged until the next correct data transmission as can be seen in Figure 4.11. The results from the Scoreboard for the 5 WORDS are shown in Figure 4.12.

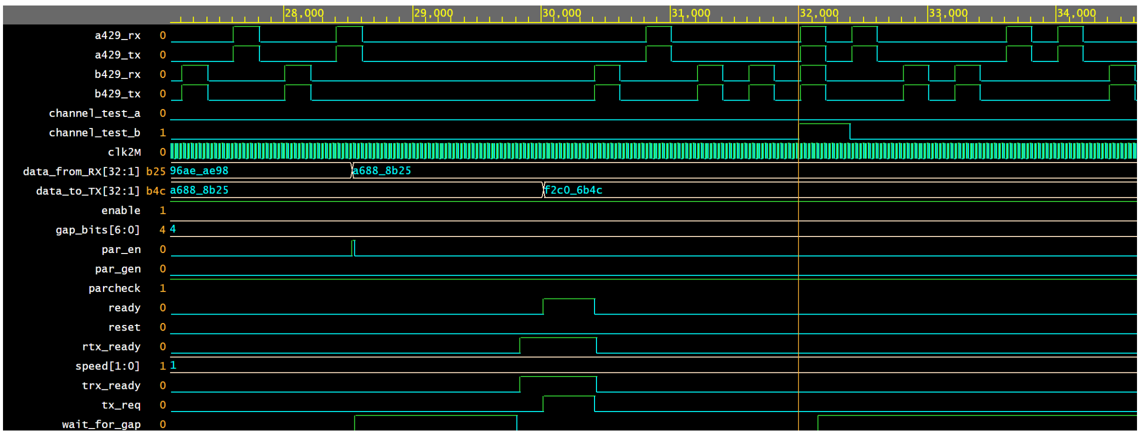


Figure 4.10 Channel's data are equal

4.2.2 Parity Bit

Parity bit is the check bit where RX checks that all the other bit's sum has an even value. If so, parity bit is set to binary 1 and the WORD is valid according to the ARINC-429 protocol. There may be a situation where each of the channels have a value that is opposite that they should have.

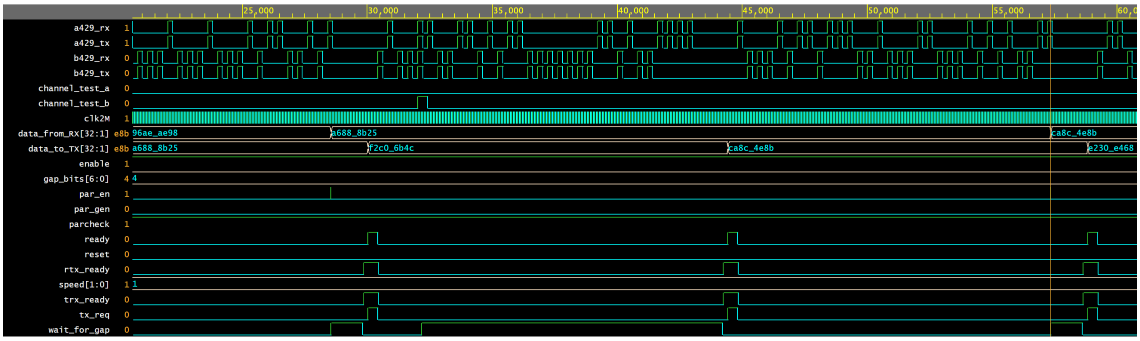


Figure 4.11 RX drops the WORD (Dropped data: F2C06B4C)

```
# KERNEL: UVM_ERROR /home/runner/scoreboard_a429.svh(62) @ 32150: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data a6888b25 does not match expected f2c06b4c
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 57350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 57350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = ca8c4e8b matched with expected data = ca8c4e8b
# KERNEL: UVM_ERROR /home/runner/scoreboard_a429.svh(62) @ 67750: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data ca8c4e8b does not match expected e230e468
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 86150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 86150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = 86b74d1b matched with expected data = 86b74d1b
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 100550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 100550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = c20fb7f0 matched with expected data = c20fb7f0
# KERNEL: UVM_ERROR /home/runner/scoreboard_a429.svh(62) @ 102950: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data c20fb7f0 does not match expected db9cbe6b
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 129350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 129350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = e3414cc4 matched with expected data = e3414cc4
# KERNEL: UVM_ERROR /home/runner/scoreboard_a429.svh(62) @ 138550: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data e3414cc4 does not match expected 9e747ef3
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 158150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 158150: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = db3dad64 matched with expected data = db3dad64
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 172550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 172550: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = 877e839f matched with expected data = 877e839f
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 186950: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 186950: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = deeb58d4 matched with expected data = deeb58d4
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(58) @ 201350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] Test: PASSED!
# KERNEL: UVM_INFO /home/runner/scoreboard_a429.svh(59) @ 201350: uvm_test_top.base_test_environment.environment_scoreboard [COMPARE] RX data is = f77ca22d matched with expected data = f77ca22d
# KERNEL: UVM_ERROR /home/runner/scoreboard_a429.svh(62) @ 209350: uvm_test_top.base_test_environment.environment_scoreboard [scoreboard_a429] Actual output data f77ca22d does not match expected ebb06faa
```

Figure 4.12 Scoreboard result

For example, channel A has a binary 1 and because of that channel B should has a binary 0 at the same time at some part of a data transmission. Let us assume, during this moment, channel A has binary 0 instead of binary 1 and channel B has binary 1 instead of binary 0 because of a cable error. Let us assume that this situation happened twice in a same WORD transmission. Two binary 1s are changed to binary 0 at the channel A and two binary 0s are changed to binary 1 at the channel B in parallel. Because that they changed their value at the same time, RX couldn't detect any errors as RX checks that whether they have opposite values from each other. Additionally, because that two binary 1s are changed to 0, mathematically the sum of the 31 bits are even again and it is observed that RX sets parity bit to high which leads the error condition where RX receives the wrong data. This situation is shown in Figure 4.13.

As can be seen in Figure 4.13, the channel's values are changed when "channel_test_a" and "channel_test_b" signals go high. Yet the parity bit goes to high and the wrong data is sent to the RX. Expected data was hexadecimal DB9CBE6B where the RX receives hexadecimal DB9CB86B.

Normal data transmission without changing channels values is shown in Figure 4.14.

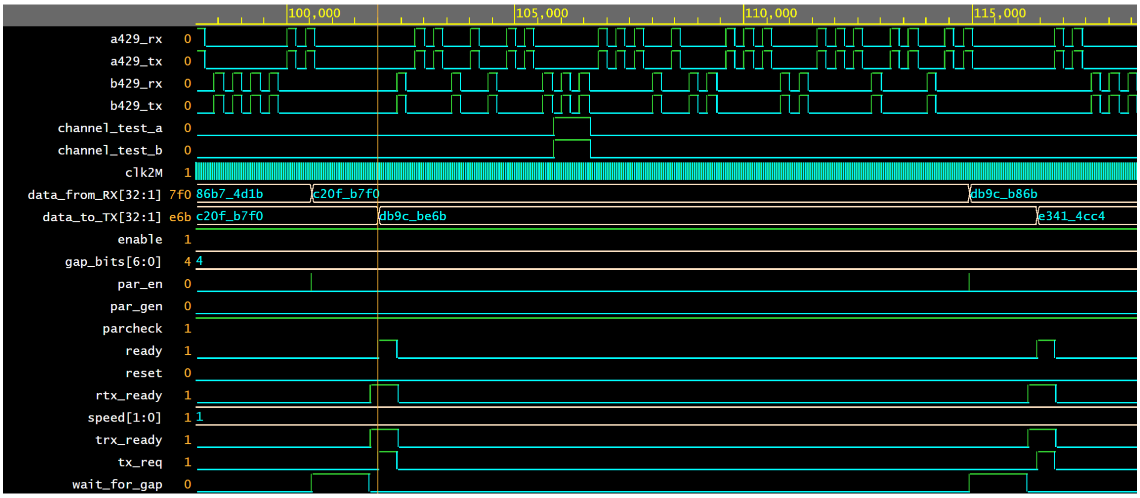


Figure 4.13 Parity fail condition

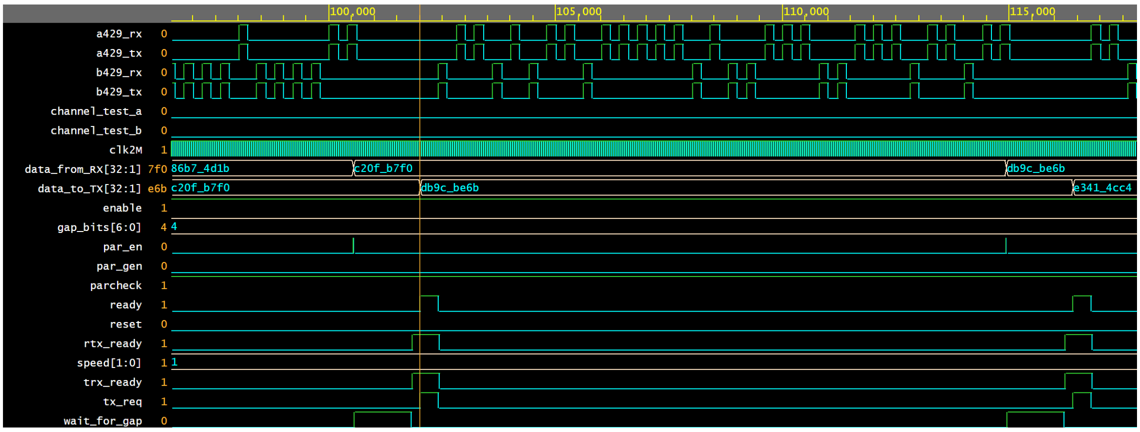


Figure 4.14 Normal data transmission

4.3 Conclusion

Within the scope of this thesis, what processes a participant who wants to design electronic equipment in the civil aviation industry should go through and how the Verification Process should be operated is explained. In order to develop DAL A and B designs, all of the DO-254 processes and the extra verification methods that explained in the Appendix B of the DO-254 should be applied. The automated simulation method is the most chosen method for PLD verification by the industry. Therefore, not only the theoretical knowledge of the work was explained, but a use case study was carried out with the usage of the UVM on the ARINC-429 design, which is the most used data bus in aviation.

First of all, the requirements were determined to start the use case study. Then, assertions were written according to the requirements. Then, an automated test environment was established with the use of the UVM. With the determination of the verification scenario and the constraint random stimuli, 1024 ARINC-429 WORDs were sent and read from the design under test. Compliance with the requirements was observed through assertions during the whole simulation. Results were supported with functional coverage and code coverage in addition to the assertions.

It was observed that the design worked correctly, but for the sake of simplicity, these observed results were implemented for a single configuration of ARINC-429. In order for a verification case to be complete, all configurations of the design must be verified under much more detailed scenarios. Moreover, all documents included in the verification case should be kept under control with configuration management tools from the very beginning to the very end of the process. At certain stages of the project, the work should be reviewed by the Quality engineer.

REFERENCES

[1] "Ieee standard for systemverilog–unified hardware design, specification, and verification language," *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pp. 1–1315, 2018. DOI: 10.1109/IEEESTD.2018.8299595.

[2] *Accellera systems initiative, "standard universal verification methodology (uvm)," 2014*. [Online]. Available: https://www.accellera.org/downloads/standards/uvm (visited on 01/01/2022).

[3] "Ieee standard for universal verification methodology language reference manual," *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, pp. 1–458, 2020. DOI: 10.1109/IEEESTD.2020.9195920.

[4] SC-180, *Design Assurance Guidance for Airborne Electronic Hardware*. RTCA Inc, 2000.

[5] SC-167, *Software Considerations in Airborne Systems and Equipment Certification*. RTCA Inc, 1992.

[6] *Radio technical committee for aeronautics*. [Online]. Available: https://www.rtca.org/ (visited on 01/01/2022).

[7] *Federal aviation administration*. [Online]. Available: https://www.faa.gov/ (visited on 01/01/2022).

[8] *Eurocae*. [Online]. Available: https://www.eurocae.net/ (visited on 01/01/2022).

[9] WG-46, *Design Assurance Guidance for Airborne Electronic Hardware*. EUROCAE, 2000.

[10] *Easa*. [Online]. Available: https://www.easa.europa.eu/ (visited on 01/01/2022).

[11] EASA, *General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances*. EASA, 2021.

[12] C. Ebert, "Requirements engineering: Management," in *Encyclopedia of Software Engineering*, P. A. Laplante, Ed., Taylor & Francis, 2010, pp. 932–948. [Online]. Available: http://www.crcnetbase.com/doi/abs/10.1081/E-ESE-120044248 (visited on 01/01/2022).

[13] R. Fulton, R. Vandermolen, *Airborne Electronic Hardware Design Assurance: A Practitioner's Guide to RTCA/DO-254*. CRC Press, 2017.

[14] S. of Automotive Engineers Aerospace, *Guidelines for Development of Civil Aircraft and Systems: SAE ARP 4754 Rev. A.* SAE, 2010.

[15] S. E. S. A. M. L. S. A. Sp, *Guidelines and methods for conducting the safety assessment process on airborne systems and equipments*. SAE, 1996.

[16] *Do-254 explained, cadence*. [Online]. Available: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/solutions/aerospace-and-defense/do-254-explained-wp.pdf (visited on 01/01/2022).

[17] D. Landoll, S. C. Beland, "Using assertions to satisfy do-254 elemental analysis," in *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, IEEE, 2011, pp. 7C4–1.

[18] *Core429 v3.7 handbook*. [Online]. Available: http://www.actel.com/ipdocs/CORE429_HB_37.pdf (visited on 01/01/2022).

[19] M. Nanda, J. Jayanthi, P. R. Rao, "Aerospace compliant test bench to verify critical aerospace functionalities," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, 2018, pp. 1500–1506.

[20] *Arinc 429 transmitter and receiver, h. alonso*. [Online]. Available: https://opencores.org/projects/a429_transmitter_receiver (visited on 01/01/2022).

[21] *Eda playgrounds*. [Online]. Available: https://www.edaplayground.com/ (visited on 01/01/2022).

[22] B. Kashyap, V. Ravi, "Universal verification methodology based verification of uart protocol," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1716, 2020, p. 012 040.

[23] *Tümdevre tasarım ve eğitim laboratuvarı*. [Online]. Available: https://tutel.bilgem.tubitak.gov.tr/home (visited on 01/01/2022).

[24] *Uvm introduction*, ChipVerify. [Online]. Available: https://www.chipverify.com/uvm/uvm-introduction (visited on 01/01/2022).

[25] *Code coverage fundamentals*. [Online]. Available: https://vlsi.pro/code-coverage-fundamentals/ (visited on 01/01/2022).

A
APPENDIX LIST

1. A. APPENDIX LIST
2. B. ASSERTIONS

B ASSERTIONS

```
// REQUIREMENTS AND ASSERTIONS

// TRANSITION REQUIREMENTS

/*
1.  ready signal shall be asserted after the wait_for_gap signal being deasserted.
*/

property ready_1;
    @(posedge clk) disable iff (reset)
  (!$isunknown(data_from_RX)) && ($fell(wait_for_gap)) |=> ##[0:$]($rose(ready));
endproperty: ready_1

A_ready_1: assert property (ready_1);
C_ready_1: cover property (ready_1);

/*
2.  Ready shall not be asserted if the enable is low.
*/

property ready_2;
    @(posedge clk) disable iff (reset)
    (!enable) |-> (!ready);
endproperty: ready_2

A_ready_2: assert property (ready_2);
C_ready_2: cover property (ready_2);

/*
3.  tx_req shall be asserted in the same clock cycle after the ready being asserted.
*/

property tx_req_1;
    @(posedge clk) disable iff (reset)
    ($rose(ready)) |-> $rose(tx_req);
endproperty: tx_req_1

A_tx_req_1: assert property (tx_req_1);
C_tx_req_1: cover property (tx_req_1);
```

Figure B.1 Assertions 1-2-3

SV/Verilog Design

```
/*
4.   tx_req shall not be asserted if the ready is low.
*/

property tx_req_2;
    @(posedge clk) disable iff (reset)
        (!ready) |-> (!tx_req);
endproperty: tx_req_2

A_tx_req_2: assert property (tx_req_2);
C_tx_req_2: cover property (tx_req_2);

/*
5.   data_to_TX shall be start sending data in the same clock cycle after
the tx_ req being asserted and data_to_TX shall not float until the tx_ req
being asserted again.
*/

property data_to_TX_1;
  @(posedge clk) disable iff (reset)
  (!$isunknown(data_from_RX)) && ($rose(tx_req)) |-> (!$stable(data_to_TX)) ##0
(!$isunknown(data_to_TX))[*0:$] ##0 ($rose(tx_req));
endproperty: data_to_TX_1

A_data_to_TX_1: assert property (data_to_TX_1);
C_data_to_TX_1: cover property (data_to_TX_1);  PASSED");

/*
6.   wr_en shall be asserted only if the data is valid.
*/

property wr_en_1;
    @(posedge clk) disable iff (reset)
        ($rose(par_en)) |-> ((data_from_RX[1] ^ data_from_RX[2] ^ data_from_RX[3]
^ data_from_RX[4] ^ data_from_RX[5] ^ data_from_RX[6] ^ data_from_RX[7] ^
data_from_RX[8] ^ data_from_RX[9] ^ data_from_RX[10] ^ data_from_RX[11] ^
data_from_RX[12] ^ data_from_RX[13] ^ data_from_RX[14] ^ data_from_RX[15] ^
data_from_RX[16] ^ data_from_RX[17] ^ data_from_RX[18] ^ data_from_RX[19] ^
data_from_RX[20] ^ data_from_RX[21] ^ data_from_RX[22] ^ data_from_RX[23] ^
data_from_RX[24] ^ data_from_RX[25] ^ data_from_RX[26] ^ data_from_RX[27] ^
data_from_RX[28] ^ data_from_RX[29] ^ data_from_RX[30] ^ data_from_RX[31] ^
data_from_RX[32]) == 1);
endproperty: wr_en_1

A_wr_en_1: assert property (wr_en_1);
C_wr_en_1: cover property (wr_en_1);
```

Figure B.2 Assertions 4-5-6

```
/*
7.  data_from_RX shall be start sending data in the
same clock cycle after the wr_en being asserted
*/

property data_from_RX_1;
    @(posedge clk) disable iff (reset)
        ($rose(par_en)) |-> (!$stable(data_from_RX));
endproperty: data_from_RX_1

A_data_from_RX_1: assert property (data_from_RX_1);
C_data_from_RX_1: cover property (data_from_RX_1);



// RESET REQUIREMENTS

/*
8.  After reset is asserted ready, tx_req, data_to_TX, data_from_RX and wr_en
shall be valued by 0.
*/

property reset_1;
    @(posedge clk) disable iff (!reset)
  (reset) |=> ((ready == 0) && (tx_req == 0) && (data_to_TX == 0) && (data_from_RX
== 0) && (par_en == 0) && (wait_for_gap == 0));
endproperty: reset_1

A_reset_1: assert property (reset_1);
C_reset_1: cover property (reset_1);
```

Figure B.3 Assertions 7-8

PUBLICATIONS FROM THE THESIS

Conference Papers

1. İ. Ayaz, U. Sakarya, İ. Hökelek, "Universal Verification Methodology Application of ARINC-429 for Airborne Electronic Hardware Certification", 13th International Conference on Electrical and Electronics Engineering (ELECO 2021), Bursa, Turkey, 25-27 November 2021.